SEDORIC 3.0
à NU

SEDORIC et STRATORIC
Versions 3.0 du 01/01/96

Troisième Partie (Pages 460 à 630)

André Chéramy
54, rue de Sours 28000 CHARTRES
cheramy@infobiogen.fr

Troisième Edition (1998)

© André Chéramy, 1998

Table des matières

Première Partie (pages 1-231)

Avant-propos 3
Comment lire ce livre 4
Nouveautés de la version 3.0 5

La RAM overlay 7
Analyse des commandes SEDORIC 7

Buffer 1 (BUF1) 16
Buffer 2 (BUF2) 17
Buffer 3 (BUF3) 18

BANQUE n°0 19
Initialisation SEDORIC 19
Source de la page 4 version ORIC-1 25
Source de la page 4 version ATMOS 25
Désassemblage de la page 4 SEDORIC 26

BANQUES interchangeables 30
BANQUE n°1 (adresse Cxxx**a**): RENUM, DELETE et MOVE 30
BANQUE n°2 (adresse Cxxx**b**): BACKUP 51
BANQUE n°3 (adresse Cxxx**c**): SEEK, CHANGE et MERGE 68
BANQUE n°4 (adresse Cxxx**d**): COPY 89
BANQUE n°5 (adresse Cxxx**e**): SYS, DNAME, DTRACK, TRACK, INIST, DNUM, DSYS, DKEY et VUSER 103
BANQUE n°6 (adresse Cxxx**f**): INIT 123
BANQUE n°7 (adresse Cxxx**g**): CHKSUM, EXT, PROT, STATUS, SYSTEM ,UNPROT et VISUHIRES 144

Début du NOYAU permanent de SEDORIC (#C800 à #FFFF) 161
Mots Clés SEDORIC 167

XRWTS Routine de gestion des lecteurs 179

Série d'appels à des sous-programmes en ROM 187
Routines SEDORIC d’usage général . . . . . 197, 212 et 236
Routines principales de Ray McLaughlin 292

Entrée SEDORIC: recherche l'adresse d'exécution d'un mot-clé SEDORIC 199
Analyse d'un nom de fichier 203
Prendre un caractère au clavier (remplace EB78 ROM) 221

Deuxième Partie (pages 232-459)

Commandes SEDORIC (avec quelques routines associées, d'usage général) . . . . . 232 et 251
Commandes SEDORIC faisant appel à une BANQUE externe 356

Note sur les coordonnées colonne/ligne ORIC-1 / ATMOS / SEDORIC 363

Gestion de fichiers ... 374

Table des vecteurs système (#FF43-#FFC6) ... 456

Copyrights ... 6, 146, 458, 461, 465, 484 à 486 et 488

Troisième Partie (pages 460-630)
ANNEXES ... 460

ANNEXE n° 1: SEDORIC V2.0 ... 461
ANNEXE n° 2: SEDORIC V2.0 ... 463
ANNEXE n° 3: SEDORIC V2.0 ... 465
ANNEXE n° 4: PATCH 001 ... 475
ANNEXE n° 5: PATCH 002 ... 478
ANNEXE n° 6: Que se passe t-il lors du boot ? ... 482
ANNEXE n° 7: Rappel de la structure des disquettes SEDORIC ... 484
ANNEXE n° 8: Que se passe t-il lors d'un SAVE ? ... 504
ANNEXE n° 9: Que se passe t-il lors d'un DEL ? ... 511
ANNEXE n° 10: Listing de l'EPROM du MICRODISC ... 517
ANNEXE n° 11: Le FDC 1793 ... 549
ANNEXE n° 12: F.A.Q ... 560
ANNEXE n° 13: Exercices de passage ROM <--> RAM overlay ... 562
ANNEXE n° 14: Utilisation d'une commande SEDORIC sans argument (programme LM) ... 564
ANNEXE n° 15: Utilisation d'une routine en RAM overlay (programme LM) ... 565
ANNEXE n° 16: Utilisation d'une commande SEDORIC avec paramètres (programme LM) ... 566
ANNEXE n° 17: Les bogues de SEDORIC ... 569
ANNEXE n° 18: Mots clés SEDORIC ... 575
ANNEXE n° 19: Les Codes de Fonctions ... 577
ANNEXE n° 20: Futures extensions ... 584
ANNEXE n° 21: Routines d'intérêt général (par ordre chronologique) ... 587
ANNEXE n° 22: Routines d'intérêt général (par thèmes) ... 597
ANNEXE n° 23: Des drives et des DOS pour ORIC ... 611
ANNEXE n° 24: Directories des disquettes SEDORIC V3.006 et TOOLS V3.006 ... 623
ANNEXE n° 25: Tables et figures ... 626
ANNEXE n° 26: Table des matières ... 628

ANNEXES

ANNEXE n° 1
SEDORIC V2.0

Cette version est due à Ray McLaughlin. D'une part elle corrige certaines bogues, d'autre part elle permet d'utiliser des disquettes 3"1/2 double densité au maximum de leurs possibilités soit 720 kilo octets (Double face, 80 pistes de 18 secteurs) et même un peu plus (747 kilo octets avec 83 pistes de 18 secteurs). Mais, comme avec la version 1.006, le formatage en 19 secteurs par piste n'est pas fiable. Il faudrait essayer avec un lecteur de type HD (1.44 Mo), mais les disquettes sont chères pour un gain de capacité minime.

La grande nouveauté consiste à disposer d'un deuxième secteur de bitmap que Ray a placé dans le troisième secteur de la piste 2O, qui était déjà réservé, mais inutilisé. On peut tirer un grand coup de chapeau à notre ami Ray, car il a fait du beau travail!

A l'usage, cette version 2.0 se révèle très pratique, même si l'on s'en tient à l'utilisation des lecteurs 3" et 5"1/4 (bogue double face corrigée). De plus, les lecteurs 3"1/2 étant bon marché (150F) et d'une bien meilleure qualité que les anciens lecteurs 3", il est recommandé de passer à ce format (disquettes à 2F au lieu de 21F) et de disposer enfin de 720 kilo octets avec SEDORIC. La gestion des fichiers est très fiable et sans problème, jusqu'à 83 pistes de 18 secteurs. La commande DEL fonctionne sans bogue et je l'ai triturée dans tous les sens!

Voici la liste des commandes et sous-programmes qui ont été modifiés:

1) TRACK (en C446 sur la BANQUE n°5) Inutile modification de la vérification du nombre total de secteurs par disquette qui doit être < 3840, mais ne peut jamais dépasser 3762!

2) INIT (en C404 sur la BANQUE n°6) Même inutile vérification, mise en place d'un deuxième secteur de bitmap et correction de la bogue de double face.

3) XPMAP (en DA4C, "Prend le secteur de bitmap dans BUF2") Adaptation pour fonctionner avec 2 secteurs de bitmap, nouveau sous-programme FF43 appelé en DA4C.

4) XSMAP (en DA8A, "Sauve le secteur de bitmap sur la disquette") Adaptation pour la même raison, remplacé par le sous-programme DC80 placé au début de XSMAP. Ce sous-programme DC80 a été implanté dans un vide laissé par la modification du sous-programme DC7D (voir plus loin). Si le b7 de 2F est à 1, le deuxième secteur de bitmap présent dans BUF2 est sauvegardé par un JSR FF4F.

5) Le sous-programme DC7D "Cherche un secteur libre" a été remplacé par le sous-programme FF94.

6) La fin du sous-programme DCD6 "Calcule à quel bit et à quel octet de la bitmap correspond le secteur AY à libérer" a été remplacé par le sous-programme FFD9 tenant compte des 2 bitmaps.

7) La commande SEDORIC ">" (en F5BA, "Affecte un champ à une variable") a aussi été modifiée en F5FE/F609.

8) Une petite série de NOP (F638/F63D) a été remplacée par un sous-programme utilisé par INIT pour insérer un appel au nouveau sous-programme FF4A qui permet de sauver la deuxième bitmap.

9) La table des vecteurs système (FF43/FFC6, qui n'était en fait pas utilisée) et une partie du copyright final (FFC7/FFF9) ont été supprimées et remplacées par des sous- programmes utilisés pour réaliser les adaptations indiquées ci-dessus:

- sous-programme FF43 utilisé pour XPMAP

- sous-programme FF4A utilisé pour INIT

- sous-programme FF4F utilisé pour XSMAP

- sous-programme FF51 utilisé pour XSMAP

- sous-programme FF94 utilisé par le sous-programme "Cherche un secteur libre"

- sous-programme FFD9 utilisé par le sous-programme "A quel bit des bitmaps correspond le secteur AY à libérer"

Au total 595 bits diffèrent entre la version 1.006 et la version 2.0. Ces différences incluent aussi: 66 octets au secteur 1 de la piste 0 où le copyright: **"SEDORIC V1.006 01/01/86"** a été changé en **"SEDORIC V2.0 08/11/91 Upgraded by Ray McLaughlin to allow 80 track double sided drives."** 1 octet au secteur suivant (SEDORIC **V2.0** au lieu de **V1.0**) et bien sûr la quasi-totalité du secteur 3 de la piste 20 qui ne contenait que des zéros. De plus l'octet en C5FE à été changé (#F1 devient #2D), mais la signification de ce changement m'échappe.

La totalité des améliorations apportées avec la version 2.0 ont été gardées dans la version 3.0

ANNEXE n° 2
SEDORIC V2.1

Après comparaison des disquettes "Master" 2.0 et 2.1 j'ai trouvé que 9 secteurs sont différents: voici donc les additions et corrections que Ray a apportées à sa première mouture:

Secteur 1 de la piste 0: 40 octets différents. Correction du message de version qui devient:

"SEDORIC V2.1   22/08/93
Upgraded by Ray McLaughlin to allow
80 track double sided drives.
(D)TRACK, DNAME & INIST bugs fixed also."

(En fait, les commandes BACKUP, DKEY, DNUM et DSYS ont aussi été affectées).

Secteur 2 de la piste 0: 1 octet différent. Correction du n° de version qui devient V2.1

Secteur 5 de la piste 4: 18 octets différents. Correction de la BANQUE n°2 (BACKUP).

De C6D9 à C6EF, remplacement de la chaîne "Formating complete" par "Done", soit un gain de 14 octets qui ont permis à Ray d'insérer le code suivant à partir de C6E2:

C6E2-	A9 48	LDA #48	qui est le code de "PHA"
C6E4-	8D 15 D0	STA D015	remplace un RTS en D015
C6E7-	20 CD CF	JSR CFCD	appel de la routine XRWTS de gestion des lecteurs
C6EA-	A9 60	LDA #60	qui est le code de "RTS"
C6EC-	8D 15 D0	STA D015	remet en place le RTS d'origine
C6EF-	60	RTS	fin de la nouvelle routine

Ce nouveau sous-programme permet d'utiliser une routine XRWTS modifiée, ceci uniquement lors du positionnement de la tête et uniquement pour la commande BACKUP, sans affecter l'utilisation de XRWTS dans les autres cas. Ray pourrait-il nous éclairer sur la raison de cette modification transitoire?

Secteur 6 de la piste 4: 2 octets différents. Correction de la BANQUE n°2 (BACKUP). En C7B2, le JSR CFCD (routine XRWTS) est remplacé par JSR C6E2 (nouveau sous-programme ci-dessus).

Secteur 1 de la piste 5: 5 octets différents. Correction dans la BANQUE n°5 d'une bogue crée par Ray lors de sa modification de (D)TRACK pour la version 2.0. En C4A3 le BEQ C4D3 est remplacé par BEQ C4D1. En C4A9 le BCC C4D5 est remplacé par un BCC C4D3. En C4AD le BCS C4D5 est remplacé par un BCS C4D3. En C4D5 et C4D6 les 2 octets 20 & DE inutiles sont remplacés par 2 NOPs (EA).

Secteur 2 de la piste 5: 1 octet différent. Encore la BANQUE n°5. La bogue précédente affectait aussi la commande INIST. Pour remédier à cela, Ray a remplacé un BEQ C4D4 par un BEQ C4D2.

Secteur 3 de la piste 5: 2 octets différents. Toujours INIST, dans la BANQUE n°5, mais cette fois il s'agit d'une bogue d'origine, qui affecte également les commandes DKEY, DNAME, DNUM, DSYS & (D)TRACK. La routine C6DB, "demander la disquette cible", était boguée (mauvaise gestion de 'ESC') et a été remplacée par une nouvelle routine en C7A0 (voir plus loin). Le JSR C6DB situé en C69A est remplacé par JSR C7A0.

Secteur 4 de la piste 5: 25 octets différents. La fin de la BANQUE n°5 (de C793 à C7FF) n'était pas utilisée. Ray y a mis la nouvelle version de la routine déboguée:

C7A0-	2C 16 C0	BIT C016	teste si le b7 du flag "BANQUE changée" est à zéro
C7A3-	10 14	BPL C7B9	si oui (BANQUE pas changée), simple RTS en C7B9
C7A5-	A2 12	LDX #12	sinon (BANQUE changée), indexe le message "LOAD"
C7A7-	20 64 D3	JSR D364	et l'affiche, puis
C7AA-	20 48 D6	JSR D648	affiche "_DISC_IN_DRIVE_ "lettre du lecteur"AND_PRESS_'RETURN'" puis demande un "ESC" (C = 1) ou un "RETURN" (C = 0)
C7AD-	58	CLI	autorise les interruptions
C7AE-	90 09	BCC C7B9	simple RTS en C7B9 si 'RETURN' a été tapé
C7B0-	5 fois 68	5 fois PLA	si 'ESC', élimine 5 octets sur la pile (au lieu de 2: pour retourner à l'interpréteur, il faut retirer de la pile les adresses de retour correspondant à 2 niveaux de JSR soit 4octets, plus 1 octet mis sur la pile par un PHP).
C7B5-	20 06 D2	JSR D206	effectue un retour à la ligne (avec un JSR au lieu d'un JMP afin de pouvoir exécuter le SEC qui suit. En effet la routine D206 met C à zéro or pour témoigner d'une sortie par 'ESC' il faut avoir C = 1).
C7B8-	38	SEC	met la retenue à 1 (ce qui n'était pas fait préalablement)
C7B9-	60	RTS	et termine.

La routine C6DB est appelée par DTRACK (en C441), DNUM (en C4DD), DKEY (en C600), DSYS (en C522 & C528), DNAME (en C41F) et INIST (en C509) à travers un appel indirect pour les 3 dernières commandes.

Secteur 1 de la piste 20: 4 octets différents. Modification de la table des drives qui contient le nombre de pistes par défaut pour chaque lecteur en service. Ces valeurs dépendent du dernier utilisateur de la commande DTRACK, ce n'est peut-être pas Ray.

Bravo encore à notre ami Ray qui, avec les versions 2.0 et 2.1, a apporté une contribution majeure à l'évolution de SEDORIC. Sans lui, les disquettes 3"1/2 seraient bien moins attractives.

ANNEXE n° 3
SEDORIC V3.0

Description des changements effectués par rapport à la version 1.006 du 01/01/86

Dans le chapitre précédant, je vous ai brossé les grandes lignes de cette nouvelle version. Vous trouverez ici un résumé de la comparaison des disquettes master des versions 1.006 et 3.006. Deux patches (correctifs) ont été mis au point ultérieurement et sont décrits en ANNEXE n° 4 et 5. L'un permet d'utiliser la ligne PB5 du VIA 6522. C'est un peu trop spécialisé pour en parler maintenant. L'autre donne un supplément d'intelligence à SEDORIC: avant de demander une disquette master, il regarde s'il n'en a pas déjà une avant de délivrer le message "INSERT_MASTER_DISC_IN_DRIVE_X AND_PRESS_’RETURN’".

Lorsque vous faites appel à l'une des commandes situées dans une des BANQUES (INIT, COPY etc... **plus CHKSUM, EXT, PROT, UNPROT, STATUS, SYSTEM et VISUHIRES pour la version 3.0**), lorsque le système vous demande une disquette master, veillez à bien lui fournir une Master **de la même version que celle qui vous a servi à booter**. En cas de doute, effectuerez un DIR qui affichera "_V3_(Mst)_" ou "_V3_(Slv)_" si le boot à été effectué en V3.0, la disquette présente dans le lecteur pouvant être d'une version quelconque. **Attention**, la commande DIR ne se soucie pas de la version de la disquette placée dans le lecteur! Par contre elle indique bien s'il s'agit d'une Master (Mst) ou d'une Slave (Slv). SEDORIC V3.0 est 100% compatible avec les versions précédentes, mais possède une BANQUE supplémentaire, ce qui implique des modification en RAM overlay pour la gérer.

La disquette master a été largement enrichie, puisqu'elle compte quelques 52 fichiers. Parmi ceux-ci, SEDORIC3.FIX vous servira de mode d'emploi, notamment pour les nouvelles commandes CHKSUM et VISUHIRES.

Au chapitre des débogages, le plus important concerne probablement la routine "Prendre un caractère au clavier, qui avait résisté à Fabrice Broche. Vous pourrez utiliser pour de bon les touches de fonction, qui acceptent enfin les commandes SEDORIC y compris celles qui n'ont pas de n° de code et permettent d'accéder facilement aux caractères "ê" et "©" (voir manuel SEDORIC pages 53, 54, 55, 102 et 103). La commande KEYSAVE a été modifiée pour faciliter l'édition (ardue) des commandes pré-définies qui sont maintenant sauvées dans les fichiers *.KEY. Des tableaux (à photocopier et à placer dans votre Manuel ou près de votre ORIC) seront proposés plus loin pour indiquer les combinaisons de touches retenues pour la V3.0 (libre à vous de constituer vos propres claviers).

Les autres nouveaux débogages concernent les commandes CSAVE, EXT et LINPUT. Ce sont tous des défauts de jeunesse qui ont perduré depuis 1986! CSAVE ne fonctionnait plus dès que le système tournait sous SEDORIC, alors qu'avec un lecteur de cassette seul, il n'y avait pas de problème. La commande EXT n'effectuait pas de contrôle de validité sur le troisième caractère de la chaîne proposée en argument. Une expérience douloureuse pour les étourdis qui y plaçaient un"?"! Quant à LINPUT, son utilisation était limitée à une seule ligne. Le résultat était imprévisible dès que la fenêtre de saisie dépassait 38 caractères.

Finalement, l'utilisation des caractères minuscules pour taper les commandes SEDORIC était tellement boguée que personne ne s'y risquait, du moins volontairement. Elle est interdite avec la V3.0, en fait elle marche toujours, mais un peu plus mal encore puisque à la liste des nombreuses exceptions il faut maintenant ajouter "delete" et "using".

Les commandes précédemment déboguées par Ray McLaughlin dans ses versions 2.0 et 2.1 ont été fidèlement reportées dans la V3.0: ce sont ">", BACKUP, DKEY, DNAME, DNUM, DSYS, DTRACK, INIST, INIT et TRACK. Certaines corrections mineures ont été ajoutées dans la V3.0 pour BACKUP et INIT, notamment afin de transmettre la nouvelle BANQUE n°7 et aussi d'utiliser au maximum la double bitmap de Ray, soit 3838 secteurs par disquette (982528 octets!). Evidemment, on ne peut formater plus de 83 pistes de 19 secteurs avec un lecteur 3"1/2, mais EUPHORIC, qui travaille dans le virtuel, accepte sans broncher 101 pistes ce qui permet tout juste d'atteindre les 3838 secteurs.

Il y a 43 secteurs différents entre la version 3.006 du 01/01/96 et la version 1.006 du 01/01/86 prise comme référence. Ceci correspond à 2362 octets différents dont 1096 dans les secteurs existants, 1012 dans les 5 nouveaux secteurs de la BANQUE n°7 et 254 dans la piste 20. Ce texte ne peut évidemment présenter qu'un résumé succinct des modifications, qui seront traitées en détail plus loin dans le livre.

Nous verrons successivement les secteurs de la disquette qui ne sont pas inclus dans les fichiers système. Puis, nous passerons en revue les fichiers système qui ont été modifiés (NOYAU, BANQUE n°2, BANQUE n°5 et BANQUE n°6) et enfin la nouvelle BANQUE n°7 dans son intégralité. **Les exemples offerts proviennent d'une disquette master vierge formatée en 42 pistes de 17 secteurs simple face.**

Piste 0 secteur 1 (83 octets différents)

VERSION:
"SEDORIC V3.006 01/01/96
Upgraded by Ray McLaughlin
and André Chéramy
See SEDORIC3.FIX file for information"

Piste 0 secteur 2 (1 octet différent)

COPYRIGHT:
"SEDORIC V3.0
© 1985 ORIC INTERNATIONAL"

Piste 20 secteur 1 (4 octets différents)

TABDRV: La table de configuration des lecteurs TABDRV devient: "D2 D2 D2 D2" soit 80 pistes double face pour les lecteurs A, B, C et D.

Piste 20 secteur 2 & 3 (250 octets différents)

DOUBLE BITMAP: Répercute l'existence de la nouvelle BANQUE n°7.

MODIFICATIONS DANS LE NOYAU

(qui sera copié en RAM overlay de C400 à FFFF)

C500- **Piste 0 secteur 6** (1 octet différent)

Modification introduite par Ray (signification inconnue).

C600- **Piste 0 secteur 7 & 8** (4 octets différents)

Correction de la **BOGUE "CSAVE"**

C700- **Piste 0 secteurs 9 et 10** (189 + 173 = 362 octets différents)

TABLES KEYDEF, **REDEF et PREDEF**

La table KEYDEF a été complètement revue pour intégrer des commandes SEDORIC. Ceci a été rendu possible grâce à la correction de la routine "Prendre un caractère au clavier". Cette nouvelle table permet d'accéder aux fonctions **BASIC** avec **FUNCT+SHIFT+touche** et aux commandes **SEDORIC** avec **FUNCT+touche**. Et ceci en respectant autant que possible les initiales. Les commandes SEDORIC sans n° (UNPROT, USING, VISUHIRES, VUSER, WIDTH, WINDOW et !RESTORE) sont maintenant accessibles.

Touche	FUNCT (SEDORIC)	Cde n°	FUNCT+SHIFT (BASIC)	Token n°
A/Q	AZERTY	#22	AND	#D1
B	BACKUP	#25	NOT	#CA
C	COPY	#29	CHR$	#ED
D	**DIR**	**#31**	DATA	#91
E	ESAVE	#3D	ELSE	#C8
F	FIELD	#3F	FOR	#8D
G	CHANGE	#27	GOSUB	#9B
H	HCUR	#41	HIRES	#A2
I	INIT	#42	INPUT	#92
J	JUMP	#45	INK	#B2
K	KEYSAVE	#4B	KEY$	#F1
L	LINPUT	#52	**LIST**	**#BC**
M/?	MOVE	#57	MUSIC	#A8
N	NUM	#59	NEXT	#90
O	OLD	#5B	OR	#D2
P	PROT	#5E	PLOT	#87
Q/A	QWERTY	#62	RESTORE	#9A
R	RENUM	#66	RETURN	#9C
S	SAVEU	#72	STEP	#CB
T	TYPE	#7B	THEN	#C9
U	UNPROT	#18	UNTIL	#8C
V	VISUHIRES	#1B	VAL	#EB
W/Z	WINDOW	#1E	WAIT	#B5
X	SEEK	#6D	EXPLODE	#A4
Y	PAPER0:INK7	#07	PING	#A6
Z/W	CALL#F8D0+CR	#08	ZAP	#A5

Exemples: FUNCT+"D" affiche "DIR" (Code n°49 = #31, manuel SEDORIC page 103), tandis que FUNCT+SHIFT+"L" affiche "LIST" (Token n° 188 = #BC, manuel ATMOS page 315) (voir aussi manuel SEDORIC page 102). Les touches A/Q, M/?, Q/A, W/Z et Z/W ont une double étiquette. Ceci correspond aux claviers AZERTY/QWERTY. La touche ;/M n'est pas utilisée, il en est de même pour les touches ' , . et /. qui toutes ont reçu le code #00. Il est possible de re-definir ces touches à l'aide de la commande KEYDEF.

Les tables REDEF des fonctions re-définissables et PREDEF des fonctions pré-définies ont également été complètement modifiées. Les nouvelles fonctions peuvent être obtenues avec les combinaisons de touches suivantes:

Touche	FUNCT (Cdes re-définissables)	Cdes n°	Touche	FUNCT+SHIFT (Cdes pré-définies)	Cdes n°
0	espace (=rien)	#00	0	HEX$(	#10
1	DOKE#2F5,#	#01	1	CALL#	#11
2	DOKE#2F5,#467+CR	#02	2	TEXT	#12
3	DOKE#2F9,#	#03	3	FORI=1TO	#13
4	DOKE#2F9,#D070+CR	#04	4	LEFT$(	#14
5	DOKE#2FC,#	#05	5	MID$(	#15
6	DOKE#2FC,#461+CR	#06	6	RIGHT$(	#16
7	PAPER0:INK7+CR	#07	7	STR$(	#17
8	**CALL#F8D0+CR**	**#08**	8	UNPROT	#18
9	ê (ASCII n°126 = #7E)	#09	9	© (ASCII 96 = #60)	#19
- £	?HEX$(PEEK(#	#0A	- £	USING	#1A
= +	?HEX$(DEEK(#	#0B	**= +**	**VISUHIRES"**	**#1B**
\ |	PEEK(#	#0C	\ |	VUSER	#1C
/ ?	DEEK(#	#0D	/ ?	WIDTH	#1D
[{	POKE#	#0E	[{	WINDOW	#1E
]}	DOKE#	#0F	]}	!RESTORE	#1F

Exemple: FUNCT+"8" déclenche une régénération des caractères (c'est une commande utilisateur re-définissable avec KUSE, visualisable avec VUSER, manuel SEDORIC page 55 & 102), tandis que FUNCT+SHIFT+"=" affiche VISUHIRES" qu'il faut compléter pour déclencher l'affichage des écrans HIRES que l'on aura indiqués (nouvelle commande pré-définie n°27 = #1B).

NB: DOKE#2F5, #2F9 et #2FC sont les vecteurs de !,] et &(). Les touches ESC, CTRL, SHIFTg, ←, ↓, espace, ↑, →, FUNCT, SHIFTd, RETURN et DEL ainsi que les touches restantes (; ' , . /) reçoivent le code de re-définition #00 soit rien. FUNCT+RETURN affiche le numéro de la ligne BASIC suivante (commande NUM).

Les tables KEYDEF, PREDEF et REDEF telles qu'elles sont décrites dans la première partie de cet article sont présentes non seulement dans le NOYAU, mais aussi dans le fichier SEDORIC3N.KEY. Le fichier SEDORIC3D.KEY contient également les mêmes tables à l'exception de la table REDEF qui a été changée pour avoir:

Touche	FUNCT	Cde n°	
0	espace (=rien)	#00	pour les touches pas encore attribuées par KEYDEF
1	POKE#26A,(PEEK(#	#01	suivie de l'une des 2 commandes suivantes
2	26A)AND#FE)	#02	pour forcer le curseur à OFF (invisible)
3	26A)OR#01)	#03	pour forcer le curseur à ON (visible)
4	PRINTCHR$(18);	#04	pour valider la commande curseur ON/OFF
5	POKE#BBA3,#0	#05	pour effacer le hideux CAPS de la ligne service
6	FORI=#BB80TO#BBA	#06	suivie de la commande suivante
7	7:POKEI,32:NEXTI	#07	pour effacer toute la ligne service
8	POKE#BB80,	#08	suivie de la commande suivante
9	PEEK(#26B)	#09	pour couleur PAPER ligne service = PAPER écran
- £	POKE#BB81,	#0A	suivie de la commande suivante
= +	PEEK(#26C)	#0B	pour couleur INK ligne service = INK écran
\ |	POKE#20C,#FF	#0C	pour forcer en mode MAJUSCULE
/ ?	POKE#20C,#7F	#0D	pour forcer en mode minuscule
[{	?HEX$(PEEK(#	#0E	pour afficher le contenu hexadécimal d'un octet
]}	?HEX$(DEEK(#	#0F	pour afficher le contenu hexadécimal de 2 octets

Exemple: Vous êtes en train de taper un programme BASIC et vous voulez effacer le curseur. Au lieu de l'habituelle bascule PRINT CHR$(17), vous voulez taper POKE#26A,(PEEK(#26A)AND#FE) qui force à OFF indépendamment de l'état précédent. Pour cela, il suffit de taper FUNCT+1 puis FUNCT+2. Si nécessaire il faut ajouter un PRINTCHR$(18); pour valider la commande précédente: Tapez simplement FUNCT+4. Rappel: le tableau ci-dessus est à photocopier et à placer dans votre Manuel ou près de votre ORIC.

Pour les utilisateurs désireux de ne rien changer à leurs habitudes, le fichier SEDORIC1.KEY contient les tables KEYDEF, PREDEF et REDEF correspondant au clavier de la version 1.006.

CA00- **Piste 0 secteur 11, 12 & 13**, (37 octets différents)

TABLE DES MOTS-CLES SEDORIC, TABLE DES INITIALES, DES ADRESSES D'EXÉCUTION

Adaptation des commandes CHKSUM, DELETE, PROT, USING, UNPROT, VISUHIRES, STATUS, SYSTEM.

CF00- **Piste 0 secteur 16** (14 octets différents)

MODIFICATION DES MESSAGES

"_(Master)_" -> "_V3_(Mst)_" et "_(Slave_)_" -> "_V3_(Slv)_"

D900- **Piste 1 secteur 9** (20 octets différents)

BOGUE "LOVE" (PRENDRE UN CARACTERE AU CLAVIER)

DA00- **Piste 1 secteur 10** (8 octets différents)

ROUTINES XPMAP ET XSMAP (DOUBLE BITMAP)

DC00- **Piste 1 secteur 12** (19 octets différents)

ROUTINE "CHERCHE UN SECTEUR LIBRE"

DD00- **Piste 1 secteur 13** (4 octets différents)

GESTION BITMAP et MODIFICATION KEYSAVE

E300- **Piste 2 secteur 2 & 3** (15 + 9 = 24 octets différents)

EXTENSION "BIGDISK" (INIT)

E600- **Piste 2 secteur 5 & 6** (207 + 11 = 218 octets différents)

ROUTINES PRINCIPALES DE RAY

Elles permettent de formater les disquettes avec le double de secteurs. Faute de place, Ray avait sacrifié la table des vecteurs (de FF43 à FFF9, soit 183 octets) pour implémenter ce code. Dans la version 3.0 de SEDORIC, la table des vecteurs a été restaurée à sa place d'origine, ce qui permet de retrouver une compatibilité avec tous les programmes écrits en langage machine, utilisant les routines de SEDORIC. Ces routines ont été mises à la place des commandes STATUS, PROT, UNPROT, SYSTEM elles mêmes déplacées dans la BANQUE n°7.

E900- **Piste 2 secteurs 8 & 9** (18 octets différents)

NOUVELLES ENTREES DES COMMANDES

CHKSUM, EXT, PROT, STATUS, SYSTEM, UNPROT, VISUHIRES.

EA99- **Piste 2 secteur 9 & 11** (57 + 2 = 59 octets différents)

CORRECTION BOGUES "LOVE" et LINPUT

F100- **Piste 2 secteur 16** (1 octet différent)

NOMBRE DE SECTEURS A TRANSFERER (INIT) qui passe à #63 (99).

F500- **Piste 3 secteur 3 & 4** (2 + 10 = 12 octets différents)

CORRECTION BOGUE COMMANDE ">"

MODIFICATIONS DANS LA BANQUE n°2

(qui sera copié en RAM overlay de C400 à C7FF)

C600- **Piste 4 secteur 5 & 6** (20 octets différents)

MODIFICATION DE LA COMMANDE BACKUP

Le message "Formating complete" a été raccourci en "Done" par Ray, ce qui permet de dégager 14 octets pour insérer un sous-programme de débogage.

STRATORIC V3.0: Cette modification n'est pas supportée par STRATORIC V3.0 qui plante sans raison apparente. Elle a donc du être neutralisée. En outre, STRATORIC comporte 2 octets différents de SEDORIC, qui influent sur les caractéristiques de formatage. Lorsque l'on veut effectuer un BACKUP avec STRATORIC V3.0, il faut donc impérativement utiliser une disquette master STRATORIC V3.0 ou V1.0.

MODIFICATIONS DANS LA BANQUE n°5

(qui sera copié en RAM overlay de C400 à C7FF)

C400- **Piste 5 secteur 1 & 2** (53 octets différents)

EXTENSION "BIGDISK" et NETTOYAGE (D)TRACK

C600- **Piste 5 secteur 3 & 4** (110 octets différents)

CORRECTION D'UNE BOGUE DE LA BANQUE n°5

Elle affectait les commandes **DKEY**, **DNAME**, **DNUM**, **DSYS**, **DTRACK**, **INIST** & **TRACK** .

MODIFICATIONS DANS LA BANQUE n°6 (C400 à C7FF)

(qui sera copié en RAM overlay de C400 à C7FF)

C400- **Piste 5 secteur 6** (1 octet différent)

EXTENSION "BIGDISK" (INIT)

Maximum 101 pistes par face au lieu de 99, utilisable avec EUPHORIC, les lecteurs 3"1/2 restants quant à eux limités à 82 pistes par face.
.

C500- **Piste 5 secteur 7** (15 octets différents)

COMMANDE INIT (DOUBLE BITMAP et "BIGDISK" suite**)**

C600- **Piste 5 secteur 8** (11 octets différents)

COMMANDE INIT (DOUBLE BITMAP suite **et "la" BOGUE)**

C700- **Piste 5 secteur 9** (1 octet différent)

CORRECTION BOGUE DE LA COMMANDE INIT (suite & fin)

STRATORIC V3.0: STRATORIC comporte ici aussi 2 octets différents de SEDORIC, qui influent sur les caractéristiques de formatage.

NOUVELLE BANQUE n°7

(qui sera copié en RAM overlay de C400 à C7FF)

Piste 5 secteur 10 (nouveau)

DESCRIPTEUR DE LA BANQUE n°7:

```
0000 00 00 FF 40 00 C4 FF C7 00 00 04 00 05 0B 05 0C
0010 05 0D 05 0E 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 etc...
```

Les 14 octets qui différent de ceux d'un secteur vierge sont indiqués en gras

C400- **Piste 5 secteur 11 à 14** (nouveaux)

BANQUE n°7 PROPREMENT DITE

Nouvel emplacement des commandes **EXT, PROT, STATUS, SYSTEM et UNPROT**. Nouvelles commandes **CHKSUM** et **VISUHIRES**.

Dans ces 4 secteurs, 998 octets (247 + 249 + 251 + 251) différent de ceux d'un secteur vierge.

ANNEXE n° 4
Patch.001

"INSERT MASTER DISK IN DRIVE..."

La capacité des disquettes 3"1/2 est énorme en comparaison de la taille moyenne des fichiers. Terminé la pénurie de place. Bonjour le fouillis des listings de directory (à ce propos, la disquette MASTER de SEDORIC V3.0 est livrée avec une version améliorée de l'utilitaire de mise en ordre alphabétique). Mais en pratique, la commande DIR de SEDORIC est devenue peu pratique. En fait, comme l'a souligné Fabrice Francès, ce qui manque maintenant, c'est une gestion de sous-répertoires. C'est dire que l'utilisation de disquettes SLAVE, bien que possible, est obsolète et je ne la recommande pas et à moins d'être masochiste.

Donc avec SEDORIC 3.0, je vous recommande de laisser une disquette MASTER en permanence dans votre drive système. Toutefois, même dans ce cas, on est rapidement excédé par le message "**INSERT_MASTER_DISK_IN_DRIVE_A AND_PRESS_'RETURN'**" à chaque fois que SEDORIC veut changer de BANQUE. Et là, je doit reconnaître ma culpabilité, puisque la version 3.0 comporte 7 BANQUES au lieu de 6.

C'est Laurent qui a attiré mon attention sur ce problème. Comment rendre SEDORIC assez intelligent pour vérifier la présence d'une disquette "Master" dans le drive système avant de réclamer ce qu'il a probablement déjà? Et si, dans l'affirmative, il chargeait la BANQUE requise sans rien dire, comme un grand? Mais SEDORIC devrait vérifier que la "Master" utilisée est bien une V3.0 (ceci est nécessaire puisque certaines commandes sont passées dans la nouvelle BANQUE n°7).

Nature du problème

Comme vous l'avez sans doute compris, à chaque commande se trouvant dans une BANQUE, correspond une paire de valeurs X et Y. X est la position de la BANQUE sur la disquette et Y est le LL (octet de poids faible) de l'adresse d'exécution de cette commande dans la première page de cette BANQUE.

L'entrée réelle de la routine de gestion des changements de BANQUE se trouve en F15E (RAM overlay). SEDORIC examine si la BANQUE demandée est déjà en place. Si ce n'est pas le cas (ou si la commande demandée est INIT), SEDORIC réclame une "Master" sans chercher à savoir s'il l'a déjà sous la main ou non.

Solution

Vous trouverez ci-dessous les indications nécessaires pour faire vous-même le patch.001 qui lancé lors du boot grâce à la commande INIST corrigera automatiquement SEDORIC 3.0. Vous pouvez également vous procurer le fichier prêt à utiliser en vous adressant CEO ou à moi-même. Si une version 3.1 de SEDORIC voit le jour, ce correctif y sera évidemment intégré.

Dans ce patch.001, j'ai remplacé quelques octets de la routine incriminée par un JMP vers une routine corrective. Cette nouvelle routine examine si la disquette présente dans le drive système est une "Master" V3.0. Si la disquette est bien une "Master" V3.0, SEDORIC reprendra le cours normal de la gestion des BANQUES, sans vous importuner. Si la disquette présente n'est pas une "Master" V3.0, SEDORIC reprendra le cours des choses avec un "INSERT_MASTER..." S'il n'y a aucune disquette, SEDORIC fera tourner le drive jusqu'à ce que vous en insériez une.

Mise en pratique...

Il vous suffira de vous reporter dans ce livre aux adresses indiquées pour comprendre les modifications apportées. Le fichier patch.001 est un fichier “mergé” composé de 3 éléments. Pour l'élaborer, procédez comme suit. Bootez avec une disquette SEDORIC V3.0. Tapez HIMEM#1F77. Utilisez soit votre moniteur favori, soit votre courage pour POKEr la suite.

1) Le premier élément "P1" est formé de 5 octets situé en RAM overlay de F16D à F171 pour la dérivation vers la routine corrective. Il sera bâti en RAM de 416D à 4171. Pour cela, vous pouvez au choix POKEr les 5 octets: EA EA 4C E5 E6 de 416D à 4171 ou assembler le code suivant:

```
416D-   EA         NOP
416E-   EA         NOP
416F-   4C E5 E6   JMP E6E5   saut vers la routine corrective
```

Puis tapez SAVE"P1",A#416D,E#4171 suivit de STATUS"P1",A#F16D. Voilà, pour la première correction!

2) Le deuxième élément "P2" comporte 38 octets de E6E5 à E70A (en RAM de 36E5 à 370A) pour la routine corrective elle-même. POKEz les 38 octets suivants:

EA	EA	AD	0A	C0	8D	00	C0	A9	00
A0	02	20	60	DA	AE	16	C2	D0	07
AE	DA	C2	E0	33	F0	08	A2	0C	20
6C	D3	4C	72	F1	4C	8F	F1		

Ou assembler le code suivant, de 36E5 à 370A:

```
36E5-   EA         NOP
36E6-   EA         NOP
36E7-   AD 0A C0   LDA C00A   le drive système
36EA-   8D 00 C0   STA C000   devient le drive actif
36ED-   A9 00      LDA $00    piste n°00
36EF-   A0 02      LDY $02    secteur n°02
36F1-   20 60 DA   JSR DA60   XPBUF2 charge dans BUF2 le secteur Y de la piste A
36F4-   AE 16 C2   LDX C216   drapeau Master/Slave
36F7-   D0 07      BNE 3700   c'est pas une Master
36F9-   AE DA C2   LDX C2DA   n° de version
36FC-   E0 33      CPX $33    est-ce "3"
36FE-   F0 08      BEQ 3708   c'est le cas, sinon...
3700-   A2 0C      LDX $0C    restaure les 5 octets
```

3702-	20 6C D3	JSR D36C	d'origine (voir en 416D)
3705-	4C 72 F1	JMP F172	et demande Master
3708-	4C 8F F1	JMP F18F	reprend sans demander

Puis tapez SAVE"P2",A#36E5,E#370A suivit de STATUS"P2",A#E6E5. Le plus gros morceau est terminé.

3) Le troisième et dernier élément "P3" est constitué de 5 octets de CF78 à CF7C (en RAM de 1F78 à 1F7C) pour la modification du fameux message: "INSERT_MASTER..." et que vous changerez en "INSERT_M**st_V3**" ce qui correspond aux 5 octets suivants: **73 74 20 56 33** à POKEr de 1F78 à 1F7C ou à entrer à l'aide de votre moniteur. Puis tapez SAVE"P3",A#1F78,E#1F7C suivi de STATUS"P3",A#CF78

Enfin, tapez COPYM"P?"TO"PATCH.001" pour rassembler les trois corrections dans le même fichier. Vérifiez votre travail avec CHKSUM"PATCH.001" qui doit vous donner les indications suivantes:

PATCH.001	F16D	F171	40	0000	03EB
PATCH.001	E6E5	E70A	40	0000	1377
PATCH.001	CF78	CF7C	40	0000	0190

Si les checksums obtenues sont différentes, vous avez fait une erreur en POKEant.

Enfin lancez INIST et ajouter PATCH.001 aux commandes initiales. Lorsque vous booterez avec cette disquette, le NOYAU situé en RAM overlay de C400 à FFFF sera automatiquement corrigé. SEDORIC V3.0 lui-même n'est pas affecté par cette procédure.

Re-bootez et testez. Par exemple, PROT protège vos fichiers sans rien demander, bien que la commande PROT soit dans la BANQUE n°7. Attention, ma modification est même brutale, car contrairement à ce qui se passait avant, nous n'avons plus a taper 'RETURN'... ni 'ESC'!

Conclusion...

Comme vous le verrez, le fonctionnement de SEDORIC est considérablement amélioré par cette petite modification de rien du tout. L'appel aux BANQUES devient complètement transparent. SEDORIC utilise maintenant 48 (RAM) + 16 (ROM) + 16 (RAM overlay) + 7 (BANQUES) = 87 koctets sans que vous le remarquiez! Ce "patch" marche aussi avec le kit STRATORIC.

ANNEXE n° 5
Patch.002

Correction de la gestion de PB5

Notre ATMOS est décidément une petite machine merveilleuse, simple et ouverte: Il est possible d'en comprendre chaque détail. A force de scruter cette petite chose, nombre d'entre nous ont pu constater qu'une des pattes du VIA 6522 est non connectée d'origine. Il s'agit de la ligne PB5 (Port B, bit n°5). C'est bête de laisser non utilisée quelque chose d'aussi précieux qu'une ligne d'entrée/sortie! Encore faut-il que SEDORIC sache respecter cette ligne PB5 dont il n'avait que faire jusqu'ici. Nous allons lui apprendre.

Nature du problème

PB5 n'ayant jamais été connecté, personne ne s'en est soucié. Résultat, beaucoup de programmes, massacrent PB5. Je veux dire que lorsqu'un programme écrit sur le Port B, il modifie de manière erratique l'état de PB5. Pour ceux qui voudraient enfin utiliser PB5, il devient donc nécessaire de disposer d'une recette universelle pour corriger les programmes existants, y compris SEDORIC.

Rappelons que le Port A est utilisé pour l'imprimante, le son et le clavier. Le Port B est impliqué dans d'autres tâches: PB0 à PB3 pour le clavier, PB4 pour le STROBE de l'imprimante, PB6 pour le "Remote control" du lecteur de K7 et enfin PB7 pour l'entrée/sortie des data K7. PB5 est resté inutilisé. Deux registres sont utilisés pour chaque port: un registre de direction des échanges (entrée ou sortie) et un registre de data (là où il faut lire ou écrire sur le port). Pour le Port B, ces registres sont respectivement accessibles aux adresses #0302 et #300. En fait, à chacune des 8 lignes d'un port correspond un bit dans ces registres. Par exemple pour mettre PB5 en sortie il faut poker #20 (00**1**0 0000) en #0302. Et pour tirer PB5 au +5V (haut logique), il faut poker #20 en #0300. En pratique, ce n'est pas si simple, car il ne faut toucher qu'au bit n°5 (la numérotation commence au bit n°0). Dans l'exemple ci-dessus, nous avons non seulement mis PB5 en sortie, mais aussi forcé les autres lignes en entrée! Le protocole à utiliser pour programmer correctement est indiqué plus loin.

Principe de la correction

La commande fautive étant toujours un STA 0302 (qui occupe 3 octets), il suffira: 1) De la remplacer dans le code à corriger par un JSR XXXX (qui occupe lui aussi 3 octets). 2) D'installer à l'adresse XXXX une petite routine qui lira le contenu du registre #0300, modifiera le bit n°5 sans changer la valeur des autres bits et re-écrira le résultat en #0300.

Correction de SEDORIC

C'est bien sûr par là qu'il faut commencer. Toutes les versions de SEDORIC sont affectées, mais je ne corrigerai que la version 3.0. Je vous propose de fabriquer une petite rustine, le PATCH.002, qui viendra se coller sur la partie fautive de SEDORIC, en RAM overlay. Comme précédemment avec le PATCH.001

qui corrigeait le "INSERT_MASTER_DISC_IN_DRIVE...", il faudra insérer PATCH.002 dans la commande INIST, afin que la correction prenne effet dès le boot. Attention, notez que seule la RAM overlay sera modifiée et non votre disquette master SEDORIC.

Vous allez assembler (ou poker directement les octets indiqués ci-dessous, pour ceux qui n'ont pas d'assembleur) les deux modifications en RAM, les sauver, changer les adresses des 2 fichiers sauvés avec la commande STATUS (pour qu'ils soient ensuite chargés directement à la bonne place dans la RAM overlay) et enfin les merger dans le fichier PATCH.002

1) Assemblez (ou pokez) la routine corrective à l'adresse 981E:

981E-	48	PHA	sauve la valeur "**V**" qui était destinée au Port B
981F-	A9 20	LDA #20	soit masque 00**1**0 0000 pour lire l'état actuel "**X**" de PB5 en 0300
9821-	2D 00 03	AND 0300	résultat: l'accumulateur contient maintenant 00**X**0 0000
9824-	8D **2B EA**	STA **EA2B**	qui met à jour le **#00** dans la routine elle-même (un peu plus loin)
9827-	68	PLA	récupère la valeur "**V**" d'origine à écrire dans le Port B
9828-	29 DF	AND #DF	soit le masque 11**0**1 1111 qui force à 0 le bit 5 de "**V**" puis le
982A-	09 **00**	ORA **#00**	remplace par le bit 5 d'origine en gardant les autres bits de "**V**"
982C-	8D 00 03	STA 0300	et enfin écrit le résultat dans le Port B
982F-	60	RTS	avant de retourner au point d'appel

SAVE"P1",A#981E,E#982F puis STATUS"P1",A#EA1E et enfin CHKSUM"P1" qui doit vous indiquer les adresses en RAM overlay (EA1E à EA2F) et la CHKSUM #054C. Si ce n'est pas le cas... corrigez!

2) Assemblez (ou pokez) en 983A l'appel à cette routine qui sera patchée en RAM overlay à l'endroit où se trouve le STA 0300 fautif dans SEDORIC, c'est à dire en D83A.

983A	20 1E EA	JSR EA1E	qui occupe 3 octets comme le STA 0300 qu'il remplace

SAVE"P2",A#983A,E#983C puis STATUS"P2",A#D83A et enfin CHKSUM"P2" qui doit vous indiquer les adresses en RAM overlay (D83A à D83C) et la CHKSUM #0128. Si ce n'est pas le cas... corrigez!

Si tout va bien, terminez avec COPYM "P?" TO "PATCH.002" Vérifiez éventuellement avec un CHKSUM"PATCH.002" et ajoutez PATCH.002:?"SEDORIC est patché!" à votre INIST. Voilà, désormais SEDORIC est prêt pour les nouvelles applications utilisant PB5.

Les lecteurs attentifs se rendront peut-être compte que j'ai logé la routine corrective dans une zone de SEDORIC 3.0 qui contient des NOP et qui était donc en réserve pour ce genre d'opération.

Correction des programmes BASIC et "Langage Machine"

Comme bien sûr, SEDORIC n'est pas le seul responsable et que beaucoup de programmes perturbent aussi PB5, en cas de besoin, il vous faudra rechercher le ou les STA 0300 (ou POKE#0300) et les remplacer par des JSR XXXX (ou CALL#XXXX). En RAM, à l'adresse XXXX de votre choix, devra se trouver une routine corrective analogue à celle décrite plus haut. Vous ne pouvez pas utiliser celle que vous avez patché en RAM overlay car les JSR ou les CALL de votre correctif aboutiraient en ROM.

Prenons un exemple concret. Si vous implantez la routine corrective en 981E, il faudra changer le STA EA2B en STA 982B. C'est simple suivez le listing ci-dessus et modifiez seulement

9824	8D **2B 98**	STA **982B**	qui auto-modifie la routine elle-même en RAM

Si vous optez pour un autre emplacement, il faudra ajuster le STA 982B de façon à écrire la valeur de l'accumulateur à l'endroit correspondant de votre routine.

<u>Protocoles à utiliser pour programmer correctement en BASIC (ou en LM):</u>

Pour mettre PB5 en entrée sans modifier la direction des autres lignes:

100 A=PEEK(#302)	(LDA 0302)	pour lire l'état actuel du registre de direction du Port B
110 A=A AND #DF	(AND #DF)	masque 11**0**1 1111 pour forcer PB5 à zéro
120 POKE#302,A	(STA 0302)	les autres bits resterons tels quels.

Pour mettre PB5 en sortie sans modifier la direction des autres lignes:

100 A=PEEK(#302)	(LDA 0302)	pour lire l'état actuel du registre de direction du Port B
110 A=A OR #20	(ORA #20)	masque 00**1**0 0000 pour forcer PB5 seulement à un
120 POKE#302,A	(STA 0302)	les autres bits resterons tels quels.

Lorque PB5 est en entrée, pour lire sa valeur dans le registre data du Port B:

100 A=PEEK(#300)	(LDA 0300)	pour lire l'état actuel du registre de data du Port B
110 A=A AND #20	(AND #20)	le masque 00**1**0 0000 force tous les bits à 0 sauf PB5
120 IF A=0 THEN...	(BEQ...)	A=0 lorsque PB5 est au niveau bas

Lorsque PB5 est en sortie, pour le mettre au niveau bas (à la masse):

100 A=PEEK(#300)	(LDA 0300)	pour lire l'état actuel du registre de data du Port B
110 A=A AND #DF	(AND #DF)	masque 11**0**1 1111 pour forcer PB5 à zéro
120 POKE#300,A	(STA 0300)	les autres bits resterons tels quels.

Lorsque PB5 est en sortie, pour le mettre au niveau haut (le tirer à +5V):

100 A=PEEK(#300)	(LDA 0300)	pour lire l'état actuel du registre de data du Port B
110 A=A OR #20	(ORA #20)	masque 00**1**0 0000 pour forcer PB5 à un
120 POKE#300,A	(STA 0300)	les autres bits resterons tels quels.

<u>Conclusion...</u>

Vous n'avez plus d'excuse maintenant pour ne pas développer une application originale basée sur l'exploitation de la ligne d'entrée/sortie PB5. Ce peut être la commande d'un relais pour votre train électrique (ce qui fait deux avec PB6). Ou pour commander l'allumage d'une LED. Où la détection d'un événement externe. Ce peut être aussi tout simplement l'utilisation de cartouches PB5 (voir le "Journal du Soft" n°9) qui devrait vous permettre de profiter de 16384 octets de ROM supplémentaires afin d'y installer les routines "Langage Machine" que demande le jeu que vous en train de développer! N'hésitez pas à me

contacter si vous avez besoin d'aide.

ANNEXE n° 6
Que se passe t-il lors du boot?

Informations recueillies sur oric@lyghtforce.com
(Contributions de Fabrice Francès et Ray McLaughlin)

Un MICRODISC est connecté à votre ATMOS. Vous inserez une disquette "Master" de SEDORIC V3.0 dans le lecteur. Votre système dispose potentiellement des mémoires et supports suivants: la RAM (48 koctets, de 0000 à BFFF), la RAM overlay (16 koctets, de C000 à FFFF), la ROM de l'ATMOS (16 koctets, de C000 à FFFF), la ROM du MICRODISC (8 koctets, de E000 à FFFF) et la disquette (jusqu'à 788,5 koctets pour une disquette formatée en 83 pistes de 19 secteurs, double face). Vous allumez votre alimentation et le tout démarre, affichant d'abord le copyright Tangerine, puis le copyright SEDORIC, puis le menu SEDORIC V3.0. Quels événements se sont produits au cours de ce boot?

Par construction, le microprocesseur 6502 trouve son vecteur de RESET en FFFC/FFFD. Cela signifie que lors de la mise sous tension ou d'un reset à froid, il fait un saut à l'adresse indiquée en FFFC/FFFD. Normalement, lorsqu'aucune interface n'est branchée sur le connecteur d'extension, c'est la ROM de la carte mère qui est validée. Pour l'ATMOS, en FFFC/FFFD de cette ROM se trouve l'adresse F88F. Le microprocesseur 6502 exécute alors le code qu'il trouve à cette adresse, c'est à dire la routine COLDSTART.

Si un MICRODISC est branché sur le connecteur d'extension, quand le système est mis sous tension, la carte du MICRODISC désactive la ROM de l'ORIC-1/ATMOS, en mettant la ligne ~~ROMDIS~~ du connecteur d'extension à la masse et valide sa propre ROM. Il fait alors un saut à l'adresse qu'il trouve en FFFC/FFFD, c'est à dire en EB7E.

En fait, les premiers ORIC-1 sont sortis avec leur ROM sous forme de deux EPROMs de 8koctets et malheureusement le signal ~~ROMDIS~~ n'étant connecté qu'à une seule de ces EPROMs (en l'occurrence IC9, qui correspond à la moitié haute de la ROM) seules les adresses de E000 à FFFF de la ROM de ces premiers ORIC-1 étaient inactivées. Une ROM de 8Koctets a donc été utilisée dans l'interface du MICRODISC pour raison de compatibilité avec tous les ORIC-1/ATMOS. Dans les machines plus récentes où l'on utilise une EPROM de 16Koctets pour loger la ROM de la carte mère, le ~~ROMDIS~~ inactive toute la puce et donc toute la ROM de C000 à FFFF. Si c'est la carte du MICRODISC qui a invalidé la ROM de la carte mère au profit de sa propre ROM, et comme celle-ci ne couvre que les adresses de E000 à FFFF, les adresses de C000 à DFFF restent dirigées vers la RAM overlay. En résumé on a alors les adresses de 0000 à DFFF qui sont en RAM dont la partie C000 à DFFF correspond à ce qui est couramment appelé la RAM overlay et les adresses de E000 à FFFF qui sont dans la ROM du MICRODISC.

En EB7E de la ROM du MICRODISC, se trouve donc la routine de RESET qui est en fait le programme de boot initial. Dans ce qui suit, j'utilise le terme général "DOS" (Disk Operating System) au lieu de SEDORIC, car en fait, ce système de boot est bien antérieur au SEDORIC (voir le paragraphe suivant). La routine de boot copie l'ensemble du DOS de la disquette "Master" dans la RAM. Le DOS est d'abord chargé en mémoire basse ainsi qu'une routine pour le remonter en RAM overlay. Cette routine est déclenchée ultérieurement par le programme, après dé-activation de la ROM du MICRODISC et validation

de la RAM overlay par la carte du MICRODISC. De plus, le code de communication entre le DOS (situé en RAM overlay) et la ROM de la carte mère est copié dans la page 4 de la RAM. Puis une routine du DOS effectue l'initialisation pour le DOS lui-même et pour le système ORIC d'origine avant de passer la main à la ROM de la carte mère qui finalement attend les entrées au clavier comme d'habitude.

La ROM du MICRODISC contient une version simplifiée de ORIC DOS version 0.6. La plupart des routines ont été éliminées, mais la ROM contient toujours la table complète des adresses de ces routines, dont un tas d'adresses nulles. Ceci résulte clairement de l'assemblage d'un programme avec des étiquettes non définies. Cette ROM contient aussi des routines inutilisées (et inutilisables), ainsi que des références à des étiquettes non définies! Il semble qu'à l'origine, les développeurs voulaient avoir ORIC DOS dans cette ROM, mais que ce DOS est devenu trop gros. De plus, il est plus difficile de mettre à jour une version en ROM qu'une version en RAM (donc chargée à partir d'une disquette).

Donc, l'ORIC DOS de la ROM "maquille" l'initialisation du BASIC de l'ORIC-1/ATMOS en modifiant les variables des pages zéro et deux et même en affichant le message de Copyright. C'est la raison pour laquelle on est trompé au boot, lorsqu'on pense que le BASIC démarre comme d'habitude et bascule de façon magique sur le code de la ROM du MICRODISC.

Mais ce n'est pas tout! Le code de la ROM du MICRODISC cherche à charger ORIC DOS à partir de la disquette. Or la structure de la disquette "Master" qui contient SEDORIC est bien différente! Si on passe sur certains détails scabreux (ORIC DOS utilise par exemple des enregistrements de taille variable dans les secteurs, enregistrements contenant leur propre adresse de chargement), le copyright est extrait d'un enregistrement factice (situé dans le secteur numéro 2 de la piste zéro) et les fichiers BOOTUP.COM et SYSTEM.COM sont cherché dans un directory factice (situé dans le troisième secteur de la piste zéro). C'est au tour de SEDORIC de tromper ORIC DOS! La façon dons les systèmes comme SEDORIC font croire à L’EPROM du MICRODISC que la disquette est une ORIC DOS est déjà tout un programme... Le genre d’horreurs nées du souci de compatibilité.

La question suivante est de savoir comment la ROM du MICRODISC est capable de charger SEDORIC, dont le système de fichiers a une structure différente et qui n'utilise pas d'enregistrements comme ORICDOS. Ceci est réalisé en bernant la ROM, en lui faisant croire qu'il y a bien une disquette ORICDOS dans le lecteur. Les trois premiers secteurs des disquettes SEDORIC servent à imiter un système de fichier ORICDOS. L'ANNEXE suivante vous montre le contenu de ces 3 secteurs. En regardant de plus près, on voit que dans le troisième secteur de la piste zéro, le fichier BOOTUP.COM est soit disant présent dans le deuxième secteur de la piste zéro. Ceci est utilisé pour charger un enregistrement qui est finallement exécuté et ce petit morceau de code est responsable du chargement de SEDORIC en RAM overlay. Bien sûr, il cela aurait été plus simple si ORICDOS chargeait un secteur de boot et l'exécutait juste après! C'est ainsi que le TELEMON du TELESTRAT opère.

ANNEXE n° 7
Rappel de la structure des disquettes SEDORIC

SEDORIC occupe 107 secteurs sur une disquette MASTER en deux groupes. Le premier groupe se trouve au début de la disquette et occupe 99 secteurs à partir du secteur n°1 de la piste n°0. Le deuxième groupe se trouve à la piste n°20 et occupe les secteurs n°1, 2, 3, 4, 7, 10, 13 et 16.

Sur une disquette SLAVE, SEDORIC occupe 8 secteurs en 2 groupes. Le 1 er groupe se trouve au début de la disquette et occupe 8 secteurs à partir du secteur n°1 de la piste n°0. Ces 8 secteurs sont identiques aux secteurs correspondants d'une disquette MASTER, sauf le 23 ème octet du 2 ème secteur qui contient #00 (Master) ou #01 (Slave). Le 2 ème groupe se trouve à la piste n°20 et occupe les secteurs n°1, 2, 3, 4, 7, 10, 13 et 16. Ces 8 secteurs ont identiques aux secteurs correspondants d'une disquette MASTER, sauf le secteur de bitmap (2 ème secteur de la piste 20) dont les octets n°#02/#03 indiquent un nombre de secteurs libres différents et l'octet n°#0A qui contient #00 (Master) ou #01 (Slave). La carte des secteurs occupés (bitmap) est bien sûr également différente! Rappel: l'utilisation d'une disquette Slave nécessite la présence de SEDORIC en RAM overlay. De plus, ce type de disquette ne permet pas d'utiliser des commandes nécessitant le chargement d'une BANQUE interchangeable. Sinon, il n'y a pas de différence.

Lors d'un INIT, les 99 premiers secteurs de la disquette master sont chargés en RAM (de #3000 à 92FF) et ceci sans considération pour la syntaxe de INIT, ce qui représente une perte de temps quand il s'agit de formater un disque SLAVE où seuls les 8 premiers secteurs sont utilisés. Après certains ajustements, 99 (Master) ou 8 (Slave) secteurs sont recopiés sur la nouvelle disquette. Il est donc prudent de reprendre la disquette Master d'origine si on veut éviter l'accumulation des erreurs.

Les 3 premiers secteurs contiennent n° de version, boot et copyright et sont listés ci-après .

Les 61 suivants sont structurés comme un fichier: 1 secteur de descripteur suivi de 60 secteurs de code qui, lors du boot, sont copiés en RAM (de #1400 à #4FFF), puis en RAM overlay (de C400 à FFFF). Ce fichier n'apparaît pas au directory.

Les 30 secteurs suivants représentent les 6 BANQUES interchangeables et sont structurés en 6 fichiers de 5 secteurs: 1 secteur de descripteur suivi de 4 secteurs de code qui sont copiés en RAM overlay (de C400 à C7FF) lors de l'appel de certaines commandes. Ces fichiers n'apparaissent pas au directory.

Pour récupérer ces fichiers cachés, il suffit de partir d'une disquette master vierge formatée en 16 secteurs par piste, de créer une série de vrais fichiers à l'aide des commandes suivantes: SAVE"NOYAU.SED",A#1400,E#4FFF SAVE"BANQUE1.SED",A#C400,E#C7FF etc idem pour BANQUE2.SED, BANQUE3.SED, BANQUE4.SED, BANQUE5.SED ET BANQUE6.SED. Puis à l'aide d'un éditeur de secteur (BDDISK par exemple), il faut remplacer les coordonnées des descripteurs de ces fichiers dans le secteur 4 de la piste 20 (1 er secteur catalogue) par les coordonnées des descripteurs des fichiers cachés. Les coordonnées des descripteurs se trouvent aux 13 ème et 14 ème octets de chaque ligne de catalogue. Il faut y écrire les coordonnées suivantes: 0004, 0401, 0406, 040B, 0410, 0505 et 050A pour une disquette master formatée en 16 secteurs par piste (voir plus loin le tableau décrivant l'emplacement de SEDORIC sur une disquette master formatée en 16 secteurs par piste). Si l'on voulait pouvoir utiliser normalement cette disquette, il faudrait poursuivre les mises à jour (directory et bitmap), mais en l'état, il est déjà possible de copier ces 7 fichiers sur une disquette normale et de les exploiter à souhait.

EMPLACEMENT DE SEDORIC SUR UNE DISQUETTE MASTER 16 SECTEURS/PISTE
(correspondance entre les adresses en RAM overlay et les secteurs d'une disquette master)

PISTE	0	1	2	3	4	5	6
SECT.							
1	**copyright**	D000	E000	F000	**BK1**	C400	C500
2	**et**	D100	E100	F100	C400	C500	C600
3	**boot**	D200	E200	F200	C500	C600	C700
4	**Desc**.	D300	E300	F300	C600	C700	**Util1**
5	C400	D400	E400	F400	C700	**BK5**	**Util2**
6	C500	D500	E500	F500	**BK2**	C400	**Util3**
7	C600	D600	E600	F600	C400	C500	**etc.**
8	C700	D700	E700	F700	C500	C600	
9	C800	D800	E800	F800	C600	C700	
10	C900	D900	E900	F900	C700	**BK6**	
11	CA00	DA00	EA00	FA00	**BK3**	C400	
12	CB00	DB00	EB00	FB00	C400	C500	
13	CC00	DC00	EC00	FC00	C500	C600	
14	CD00	DD00	ED00	FD00	C600	C700	
15	CE00	DE00	EE00	FE00	C700	**BK7**	
16	CF00	DF00	EF00	FF00	**BK4**	C400	

Copyright et boot = les trois premiers secteurs de la disquette
Desc. = "descripteur" du NOYAU SEDORIC
BK1 à **BK7** = "descripteurs" des BANQUES interchangeables 1 à 7
Util1, Util2, Util3 etc. premiers secteurs libres pour l'utilisateur

EMPLACEMENT DE SEDORIC SUR UNE DISQUETTE MASTER 17 SECTEURS/PISTE
(correspondance entre les adresses en RAM overlay et les secteurs d'une disquette master)

PISTE	0	1	2	3	4	5
SECT.						
1	**copyright**	D100	E200	F300	C700	C400
2	**et**	D200	E300	F400	**BK2**	C500
3	**boot**	D300	E400	F500	C400	C600
4	**Desc**.	D400	E500	F600	C500	C700
5	C400	D500	E600	F700	C600	**BK6**
6	C500	D600	E700	F800	C700	C400
7	C600	D700	E800	F900	**BK3**	C500
8	C700	D800	E900	FA00	C400	C600
9	C800	D900	EA00	FB00	C500	C700
10	C900	DA00	EB00	FC00	C600	**BK7**
11	CA00	DB00	EC00	FD00	C700	C400
12	CB00	DC00	ED00	FE00	**BK4**	C500
13	CC00	DD00	EE00	FF00	C400	C600
14	CD00	DE00	EF00	**BK1**	C500	C700
15	CE00	DF00	F000	C400	C600	**Util1**
16	CF00	E000	F100	C500	C700	**Util2**
17	D000	E100	F200	C600	**BK5**	**etc.**

Copyright et boot = les trois premiers secteurs de la disquette
Desc. = "descripteur" du NOYAU SEDORIC
BK1 à **BK7** = "descripteurs" des BANQUES interchangeables 1 à 7
Util1, Util2, etc. premiers secteurs libres pour l'utilisateur

EMPLACEMENT DE SEDORIC SUR UNE DISQUETTE MASTER 18 SECTEURS/PISTE
(correspondance entre les adresses en RAM overlay et les secteurs d'une disquette master)

PISTE	0	1	2	3	4	5
SECT.						
1	copyright	D200	E400	F600	C600	C400
2	et	D300	E500	F700	C700	C500
3	boot	D400	E600	F800	**BK3**	C600
4	Desc.	D500	E700	F900	C400	C700
5	C400	D600	E800	FA00	C500	**BK7**
6	C500	D700	E900	FB00	C600	C400
7	C600	D800	EA00	FC00	C700	C500
8	C700	D900	EB00	FD00	**BK4**	C600
9	C800	DA00	EC00	FE00	C400	C700
10	C900	DB00	ED00	FF00	C500	**Util1**
11	CA00	DC00	EE00	**BK1**	C600	**Util2**
12	CB00	DD00	EF00	C400	C700	**Util3**
13	CC00	DE00	F000	C500	**BK5**	**etc.**
14	CD00	DF00	F100	C600	C400	
15	CE00	E000	F200	C700	C500	
16	CF00	E100	F300	**BK2**	C600	
17	D000	E200	F400	C400	C700	
18	D100	E300	F500	C500	**BK6**	

Copyright et boot = les trois premiers secteurs de la disquette
Desc. = "descripteur" du NOYAU SEDORIC
BK1 à **BK7** = "descripteurs" des BANQUES interchangeables 1 à 7
Util1, Util2, Util3 etc. premiers secteurs libres pour l'utilisateur

Dump du premier secteur de disquette Master ou Slave (VERSION)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000-	01	00	00	00	00	00	00	00	20	20	20	20	20	20	20	20	
0010-	00	00	03	00	00	00	01	00	53	45	44	4F	52	49	43	20	
0020-	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0030-	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0040-	53	45	44	4F	52	49	43	20	56	**33**	2E	30	30	36	20	30	SEDORIC V3.006 0
0050-	31	2F	30	31	2F	**39**	36	**0D**	**0A**	**55**	**70**	**67**	**72**	**61**	**64**	**65**	1/01/96..Upgrade
0060-	**64**	20	**62**	**79**	20	**52**	**61**	**79**	20	**4D**	**63**	**4C**	**61**	**75**	**67**	**68**	d by Ray McLaugh
0070-	**6C**	**69**	**6E**	20	20	20	20	20	20	20	20	20	**0D**	**0A**	**61**	**6E**	lin ..an
0080-	**64**	20	**41**	**6E**	**64**	**72**	**7B**	20	**43**	**68**	**7B**	**72**	**61**	**6D**	**79**	20	d André Chéramy
0090-	20	20	20	20	20	20	20	20	20	20	20	20	**0D**	**0A**	**0D**	**0A**	....
00A0-	**53**	**65**	**65**	20	**53**	**45**	**44**	**4F**	**52**	**49**	**43**	**33**	**2E**	**46**	**49**	**58**	See SEDORIC3.FIX
00B0-	20	**66**	**69**	**6C**	**65**	20	**66**	**6F**	**72**	20	**69**	**6E**	**66**	**6F**	**72**	**6D**	file for inform
00C0-	**61**	**74**	**69**	**6F**	**6E**	20	**0D**	**0A**	20	20	20	20	20	20	20	20	ation ..
00D0-	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00E0-	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00F0-	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	

Cet exemple provient d'une disquette Master vierge, formatée en 42 pistes de 17 secteurs, simple face. Les 83 octets qui diffèrent de leur homologues de la version 1.006 sont indiqués en gras. En 0040, le message de VERSION devient:

"SEDORIC V3.006 01/01/96
Upgraded by Ray McLaughlin
and André Chéramy
See SEDORIC3.FIX file for information"

Dump du deuxième secteur de disquette Master ou Slave (COPYRIGHT)

```
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000-      00 00 FF 00 D0 9F D0 9F 02 B9 01 00 FF 00 00 B9
0010-      E4 B9 00 00 E6 12 00 78 A9 7F 8D 0E 03 A9 10 A0        01 si Slave
0020-      07 8D 6B 02 8C 6C 02 A9 86 8D 14 03 A9 BA A0 B9
0030-      20 1A 00 A9 84 8D 14 03 A2 02 BD FD CC 9D F7 CC
0040-      CA 10 F7 A2 37 A0 80 A9 00 18 79 00 C9 C8 D0 F9
0050-      EE 37 B9 CA D0 F3 A2 04 A8 F0 08 AD 01 B9 A8 D0
0060-      02 A2 3C 84 00 A9 7B A0 B9 8D FE FF 8C FF FF A9
0070-      05 8D 12 03 A9 85 8D 14 03 A9 88 8D 10 03 A0 00
0080-      58 AD 18 03 30 FB AD 13 03 99 00 C4 C8 4C 6C B9
0090-      A9 84 8D 14 03 68 68 68 AD 10 03 29 1C D0 D5 EE
00A0-      76 B9 EE 12 03 CA F0 1F AD 12 03 CD 00 B9 D0 C1
00B0-      A9 58 8D 10 03 A0 03 88 D0 FD AD 10 03 4A B0 FA
00C0-      A9 01 8D 12 03 D0 AA A9 C0 8D 0E 03 4C 00 C4 0C
00D0-      11 53 45 44 4F 52 49 43 20 56 33 2E 30 0A 0D 60 .SEDORIC V3.0..`
00E0-      20 31 39 38 35 20 4F 52 49 43 20 49 4E 54 45 52  1985 ORIC INTER
00F0-      4E 41 54 49 4F 4E 41 4C 0D 0A 00 00 00 00 00 00 NATIONAL........
```

Cet exemple provient d'une disquette Master vierge, formatée en 42 pistes de 17 secteurs, simple face. Le seul octet qui diffère de son homologue de la version 1.006 est indiqué en gras. En 00D1, le message de COPYRIGHT devient:

"SEDORIC V3.0
© 1985 ORIC INTERNATIONAL"

Remarquez le contenu de l'octet n°#16 qui vaut ici #00 et indique qui s'agit d'une disquette Master.

Désassemblage du deuxième secteur de disquette master

Cette routine est probablement mise en jeu lors du BOOT. Elle semble charger SEDORIC en RAM overlay. Il faudrait connaître la signification des registres d'I/O du contrôleur de disquette pour pouvoir en comprendre les détails.

0017-	78	SEI	interdit les interruptions
0018-	A9 7F	LDA #7F	A = 0111 1111
001A-	8D 0E 03	STA 030E	b7 de 030E à 0 pour interdire les interruptions
001D-	A9 10	LDA #10	
001F-	A0 07	LDY #07	
0021-	8D 6B 02	STA 026B	PAPER = #10 (noir)
0024-	8C 6C 02	STY 026C	INK = 07 (blanche)
0027-	A9 86	LDA #86	
0029-	8D 14 03	STA 0314	#86 I/O contrôleur de disquette
002C-	A9 BA	LDA #BA	
002E-	A0 B9	LDY #B9	AY = #BAB9
0030-	20 1A 00	JSR 001A	afficher la chaîne pointée par AY
0033-	A9 84	LDA #84	
0035-	8D 14 03	STA 0314	#84 I/O contrôleur de disquette
0038-	A2 02	LDX #02	pour copie de 3 octets de CCFD/CCFF en CCF7/CCF9

```
003A-   BD FD CC   LDA CCFD,X   lit un caractère de "COM" (extension par défaut)
003D-   9D F7 CC   STA CCF7,X   et le copie comme extension courante
0040-   CA         DEX          caractère précédant
0041-   10 F7      BPL 003A     reboucle en 003A tant qu'il y en a à copier
```

Temporise?

```
0043-   A2 37      LDX #37
0045-   A0 80      LDY #80
0047-   A9 00      LDA #00
0049-   18         CLC
004A-   79 00 C9   ADC C900,Y   calcule A = A + contenu de C900 + Y
004D-   C8         INY          indexe le suivant
004E-   D0 F9      BNE 0049     et reboucle en 0049 tant que Y n'est pas nul
0050-   EE 37 B9   INC B937     incrémente B937 lorsque Y passe par zéro
0053-   CA         DEX          décrémente l'index X
0054-   D0 F3      BNE 0049     et reboucle en 0049 tant que X n'est pas nul
0056-   A2 04      LDX #04
0058-   A8         TAY          teste si A est nul
0059-   F0 08      BEQ 0063     si oui, continue en 0063 avec Y = #00
005B-   AD 01 B9   LDA B901     sinon, A = B901
005E-   A8         TAY          teste si A est différent de zéro
005F-   D0 02      BNE 0063     si oui, continue en 0063 avec Y <> #00
0061-   A2 3C      LDX #3C
0063-   84 00      STY 00       écrit Y en 00
0065-   A9 7B      LDA #7B
0067-   A0 B9      LDY #B9      AY = #B97B
0069-   8D FE FF   STA FFFE
006C-   8C FF FF   STY FFFF     FFFE/FFFF = #B97B
006F-   A9 05      LDA #05
0071-   8D 12 03   STA 0312     #05 I/O contrôleur de disquette
0074-   A9 85      LDA #85
0076-   8D 14 03   STA 0314     #85 I/O contrôleur de disquette
0079-   A9 88      LDA #88
007B-   8D 10 03   STA 0310     #88 I/O contrôleur de disquette
007E-   A0 00      LDY #00      index pour écriture
0080-   58         CLI          autorise les interruptions
0081-   AD 18 03   LDA 0318     teste Ready du contrôleur de disquette
0084-   30 FB      BMI 0081     reboucle en 0081 tant que b7 n'est pas à 1
0086-   AD 13 03   LDA 0313     lecture du registre data contrôleur de disquette
0089-   99 00 C4   STA C400,Y   écriture à partir de C400
008C-   C8         INY          indexe la position suivante
008D-   4C 6C B9   JMP B96C     suite en B96C

0090-   A9 84      LDA #84
0092-   8D 14 03   STA 0314     #84 I/O contrôleur de disquette
0095-   68         PLA
0096-   68         PLA          élimine 3 octets de la pile
```

0097-	68	PLA	
0098-	AD 10 03	LDA 0310	lit octet en 0310 commande contrôleur de disquette
009B-	29 1C	AND #1C	0001 1100 force à 0 tous les bits sauf b2 b3 b4
009D-	D0 D5	BNE 0074	reboucle en 0074 si le résultat n'est pas nul
009F-	EE 76 B9	INC B976	incrémente B976
00A2-	EE 12 03	INC 0312	incrémente 0312 contrôleur de disquette
00A5-	CA	DEX	décrément X
00A6-	F0 1F	BEQ 00C7	continue en 00C7 lorsque X devient nul
00A8-	AD 12 03	LDA 0312	lit octet en 0312 contrôleur de disquette
00AB-	CD 00 B9	CMP B900	teste s'il est différent du contenu de B900
00AE-	D0 C1	BNE 0071	si oui, reboucle en 0071
00B0-	A9 58	LDA #58	
00B2-	8D 10 03	STA 0310	#58 I/O commande contrôleur de disquette
00B5-	A0 03	LDY #03	pour temporisation
00B7-	88	DEY	décrémente Y
00B8-	D0 FD	BNE 00B7	et reboucle en 00B7 jusqu'à ce qu'il soit nul
00BA-	AD 10 03	LDA 0310	lit octet en 0310 commande contrôleur de disquette
00BD-	4A	LSR	teste si le b0 de l'octet lu en 0310 est à 1
00BE-	B0 FA	BCS 00BA	si oui, reboucle jusqu'à ce qu'il passe à 0
00C0-	A9 01	LDA #01	
00C2-	8D 12 03	STA 0312	#01 I/O contrôleur de disquette
00C5-	D0 AA	BNE 0071	reprise forcée en 0071
00C7-	A9 C0	LDA #C0	1100 0000
00C9-	8D 0E 03	STA 030E	autorise les interruptions T1
00CC-	4C 00 C4	JMP C400	et continue en C400 (initialisation SEDORIC)

Dump du troisième secteur de disquette master (BOOT)

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000-  00 00 02 53 59 53 54 45 4D 44 4F 53 01 00 02 00  ...SYSTEMDOS....
0010-  02 00 00 42 4F 4F 54 55 50 43 4F 4D 00 00 00 00  ...BOOTUPCOM....
0020-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0030-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0060-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0070-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0080 à 00FF idem uniquement des zéros... ce n'est pas rentable!
```

Ce secteur n'a pas été modifié depuis la version 1.006

Dump du secteur n°1 de la piste n°#14 (n°20) (SECTEUR SYSTÈME)

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
C100-  D2 D2 D2 D2 40 64 00 0A 00 20 20 20 20 20 20 20  RRRR@d....
C110-  20 20 20 20 20 20 58 58 2F 58 58 2F 58 58 20 20        XX/XX/XX
C120-  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
C130-  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
C140-  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
C150-  20 20 20 20 20 20 20 20 20 20 00 00 00 00 00 00            ......
C160-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C170-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C180-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C190-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1A0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1B0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1C0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1D0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1E0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1F0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Cet exemple provient d'une disquette Master vierge, formatée en 42 pistes de 17 secteurs, simple face. Les 4 octets qui diffèrent de leur homologues de la version 1.006 sont indiqués en gras et concernent la table de configuration des lecteurs TABDRV (les 4 premiers octets du secteur) qui devient: "D2 D2 D2 D2" soit 82 pistes double face pour les lecteurs A, B, C et D.

Le Secteur Système (secteur 1 de la piste 20) est structuré ainsi:

C100-	octets n°00/03	table des drives, contient le nombre de pistes et de faces, ici #D2 = #52 (soit 82 pistes par face) + #80 (flag double face) pour les drives A, B, C et D.
C104-	octets n°04	type de clavier (b6=1 si ACCENT SET et b7=1 si AZERTY) ici #40 = 0100 0000, seul b6 est à 1, c'est un clavier accentué en QWERTY.
C105-	octets n°05/06	départ de RENUM (ici #0064 = 100)
C107-	octets n°07/08	"pas" de RENUM (ici #0000A = 10)
C109-	octets n°09/1D	nom de la disquette (21 octets) (ici "_____________XX/XX/XX")
C11E-	octets n°1E/59	INIST, instructions exécutées au démarrage (60 octets) (ici, aucune instruction)
C15A-	octets n°5A/FF	non utilisés (suite de #00) (l'INIST aurait pu être plus long)

Dump des secteurs n°2 et n°3 de la piste n°#14 (n°20) (BITMAP)

Exemple de secteur n°2 de la piste n°20, premier secteur de bitmap:

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C200-  FF 00 5F 02 00 00 2A 11 01 2A 00 00 00 00 00 00
C210-  00 00 00 00 00 00 00 00 00 00 00 00 F8 FF FF FF
C220-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C230-  FF FF FF FF FF FF FF FF FF FF 0F DB F6 FF FF FF
C240-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C250-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C260-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C270-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C280-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C290-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2A0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2B0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2C0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2D0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2E0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2F0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Cet exemple provient d'une disquette Master vierge, formatée en 42 pistes de 17 secteurs, simple face. Les 3 octets qui diffèrent de leur homologues de la version 1.006 sont indiqués en gras. La bitmap répercute l'existence de la nouvelle BANQUE n°7 et indique 5 secteurs libres de moins. En 0003 & 0004, le nombre de secteurs libres passe de 612 (#0264) à 607 (#02**5F**) (dans mon exemple). En 001B & 001C, 5 bits de moins sont positionnés à 1: #C0 & #FF deviennent #00 & #F8 soit en binaire: 1000000 & 11111111 qui deviennent 00000000 & 11111000 (rappel: il faut lire chaque octet de droite à gauche ce qui donne 00000011 11111111 qui devient 000000**00 000**11111 où les bits correspondant au descripteur et aux 4 secteurs de la BANQUE n°7 sont maintenant positionnés à 0 (occupés).

Le premier secteur de Bitmap (secteur 2 de la piste 20) est structuré ainsi:

C200-	octets n°00	contient toujours #FF
C201-	octets n°0	contient toujours #00
C202-	octets n°02/03	nombre de secteurs libres (ici #025F = 607)
C204-	octets n°04/05	nombre de fichiers (ici #0000 = aucun)
C206-	octets n°06	nombre de pistes/face (ici #2A = 42)
C207-	octets n°07	nombre de secteurs/piste (ici #11 = 17)
C208-	octets n°08	nombre de secteurs de directory (ici #01 = 1)
C209-	octets n°09	copie de l'octet n°06 dont le b7 est ajusté à 0 si simple face ou à 1 si double face. Ici, c'est une disquette simple face. On aurait eu #AA pour une disquette double face avec 17 secteurs par face.
C20A-	octets n°0A	#00 si Master, #01 si Slave ou #47 ("G") si Games. Ici nous sommes donc en présence d'une disquette Master.
C20B-	octets n°0B/0F	contient toujours #00 (non utilisés)
C210-	octets n°10/FF	Bitmap: chaque bit représente un secteur. Ce secteur est libre si le bit correspondant est à 1 ou occupé s'il est à 0. Les bits de chaque octet sont lus de droite à gauche (sens b0 -> b7), mais les octets sont lus de gauche à droite (sens octet n°#10 -> n°#FF).

Par exemple, on voit ici que la plupart des secteurs sont libres (octets à #FF, c'est à dire à 1111 1111). Le début de la disquette est occupé par SEDORIC. Ceci est visible par les 12 zéros situés de l'octet n°#10 à l'octet n°#1B, puis par l'octet n°1C qui vaut #F8 soit 1111 1000. SEDORIC occupe donc les 99 premiers secteurs (12 x 8 = 96 plus 3). Les secteurs suivants sont libres, jusqu'à la piste n°20 (suite de 13 fois #FF).

On a alors les octets n°#3A, 3B et 3C qui indiquent #0F, #DB et #F6. Il s'agit du codage des 17 secteurs de la piste n°20. Les 4 premiers secteurs sont répertoriés par l'octet n°3A où l'on a #0F = 0000 1111. Ceci illustre la complexité du codage: il faut lire de droite à gauche (de b0 à b7). On trouve d'abord 4 bits à 1 qui concernent la fin de la piste n°19. Puis 4 bits à 0 qui indiquent que les secteurs n°1 à 4 de la piste n°20 sont occupés (secteur TABDRV, 2 secteurs de bitmap et premier secteur de directory). L'octet suivant n°#3B vaut #DB, soit 1101 1011, ce qui se lit de droite à gauche de la manière suivante: les secteurs n°5 et 6 sont libres (les bits b0 et b1 sont à 1), le secteur n°7 est occupé (bit b2 à 0) (c'est le deuxième secteur de directory), les secteurs n°8 et 9 sont libres (bit b3 et b4 à 1), le secteur n°10 est occupé (bit b5 à 0) (c'est le troisième secteur de directory), les secteurs n°11 et 12 sont libres (bits b6 et b7 à 1). Pour l'octet n°#3C où l'on a #F6 = 1111 0110 qui indique que le secteur n° 13 est occupé (b0 à 0) (quatrième secteur de directory), que les secteurs n°14 et 15 sont libres (b1 et b2 à 1), le secteur n°16 est occupé (b3 à 0) (cinquième secteur de directory) et enfin que le secteur n°17 est libre (b4 à 1). Les 3 derniers bits (b5 à b7) qui sont à 1 indiquent que les 3 premiers secteurs de la piste suivante sont libre.

Il faut noter que les secteurs n° 7, 10, 13 et 16 de la piste n°20 sont marqués occupés, même si ce n'est pas la cas. En fait ils sont "réservés" pour le directory. Cette disposition représente une optimisation de SEDORIC: la piste n°20 se trouvait au milieu des 40 pistes des disquettes d'origine, ce qui limitait les déplacement de la tête de lecture/écriture. Enfin, SEDORIC est capable de lire/écrire un secteur sur trois de la même piste et de gérer les informations pendant que la tête passe au suivant, sans avoir besoin de faire un tour complet!

Notez que la complexité apparente de ce codage n'est pas due au fait que l'état d'occupation des secteurs de la disquette est représenté par un bit, mais au fait qu'à la suite logique séquentielle du premier secteur de la première piste au dernier secteur de la dernière piste correspond à une suite de bits à lire de droite à gauche dans une suite d'octets à lire de gauche à droite. De plus il n'y a aucune correlation entre le numéro d'un octet codant et un secteur donné et encore moins entre un bit donné et et le secteur correspondant. Il faut faire le calcul: si tel secteur est le X ème de la disquette il est codé dans le (X-1)/8 ème octet situé après l'octet n°10 du secteur de bitmap et par le bit b(r) indiqué par le reste "r" de la division. Pour SEDORIC, c'est un calcul simple, mais qui dépasse les facultés de calcul mental de l'oricien moyen.

Toutefois, si vous êtes amenés à bricoler sur des disquettes SEDORIC, formatez-les en 16 secteurs par piste, ainsi il sera plus facile d'établir une correspondance entre les secteurs de la disquette et la bitmap, puisque chaque piste sera codée par deux octets tout juste!

<u>Certains secteurs sont déjà occupés au départ</u>, ce sont:

-Les secteurs situés au début de la disquette et dont le nombre varie selon qu'il s'agit d'une disquette Master (94), Slave (8) ou de type "G" (17) (Shortsed initialisé par GAMEINIT)

-Le Secteur Système (secteur 1 de la piste 20) voir ci-dessus

-Les Secteurs Bitmap (secteurs 2 et 3 de la piste 20) voir ci-dessus

-Les secteurs de catalogues (secteurs 4, 7, 10, 13 et 16 de la piste 20) voir ci-dessus. Ils sont marqués "occupés" (réservés) mais contiennent des #00 tant qu'ils ne sont pas réellement utilisés.

Voici le deuxième secteur de bitmap (secteur n°3 de la piste n°20) correspondant à l'exemple ci-dessus:

0000- **FF** 00 **CA 02** 00 00 **2A 11 01 2A** 00 00 00 00 00 00
0010- **FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF** etc...

Cet exemple provient toujours de la même disquette Master vierge, formatée en 42 pistes de 17 secteurs, simple face. Les 247 octets qui diffèrent de leur homologues de la version 1.006 sont indiqués en gras (j'ai abrégé!). En effet à l'origine, ce secteur était "réservé" (occupé), mais non utilisé. Il était rempli de #00. Maintenant, la plupart des octets valent #FF pour indiquer (par défaut) des secteurs libres.

La première ligne est identique à celle de l'autre secteur de bitmap, à l'exception des octets n°2 et 3 qui indiquent le nombre de secteurs **totaux** (ici #02CA = 714 soit 17 x 42) au lieu du nombre de secteurs libres (ici #025F = 607 soit 714 - 607 = 107 secteurs utilisés par SEDORIC sur une disquette Master de la version 3.0).

<u>Dump du secteurs n°4 de la piste n°#14 (n°20) (DIRECTORY)</u>

<u>Exemple de Directory pour l'étude des différentes sortes de descripteurs</u>

```
Drive A V3 (Mst)              XX/XX/XX

E        .DOC 259   F        .DOC 514
D        .DOC 255   A        .DOC   3
B        .DOC   4   C        .DOC   2
G        .DOC   9   PETIFICHA.DSC   2
PETIFICHB.DSC   2   PETIFICHC.DSC   2
GROSFICHD.DSC   4   GROSFICHE.DSC   4
PETIFICHM.DSC   4   GROSFICHM.DSC   7

*250 sectors free (D/42/17) 14 Files
```

Le fichier G.DOC est formé des fichiers A, B et C “mergés”. Le fichier F.DOC est formé des fichiers D et E “mergés”. Dans les 2 cas, la taille résultante est la somme des tailles.

Ne pas tenir compte des fichiers "*.DSC". La page de catalogue correspondante (piste #04 du secteur #14) est la suivante (les coordonnées du premier descripteur de chacun des fichiers qui nous intéressent sont en gras):

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
C300-	00	00	F0	00	00	00	00	00	00	00	00	00	00	00	00	00	....
C310-	45	20	20	20	20	20	20	20	20	44	4F	43	26	**0A**	**03**	41	E DOC...@
C320-	46	20	20	20	20	20	20	20	20	44	4F	43	8B	**0E**	**02**	42	F DOC...@
C330-	44	20	20	20	20	20	20	20	20	44	4F	43	0D	**0E**	**FF**	40	D DOC...@
C340-	41	20	20	20	20	20	20	20	20	44	4F	43	05	**0A**	**03**	40	A DOC...@
C350-	42	20	20	20	20	20	20	20	20	44	4F	43	05	**0D**	**04**	40	B DOC...@
C360-	43	20	20	20	20	20	20	20	20	44	4F	43	05	**11**	**02**	40	C DOC...@
C370-	47	20	20	20	20	20	20	20	20	44	4F	43	06	**02**	**09**	40	G DOC...@
C380-	50	45	54	49	46	49	43	48	41	44	53	43	06	0B	02	40	PETIFICHADSC...@
C390-	50	45	54	49	46	49	43	48	42	44	53	43	06	0D	02	40	PETIFICHBDSC...@
C3A0-	50	45	54	49	46	49	43	48	43	44	53	43	06	0F	02	40	PETIFICHCDSC...@
C3B0-	47	52	4F	53	46	49	43	48	44	44	53	43	07	04	04	40	GROSFICHDDSC...@
C3C0-	47	52	4F	53	46	49	43	48	45	44	53	43	07	08	04	40	GROSFICHEDSC...@
C3D0-	50	45	54	49	46	49	43	48	47	44	53	43	06	11	04	40	PETIFICHGDSC...@
C3E0-	47	52	4F	53	46	49	43	48	46	44	53	43	07	0C	07	40	GROSFICHFDSC...@
C3F0-	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

<u>Un Secteur de Catalogue est structuré ainsi:</u>

C300-	octets n°00/01	coordonnées (piste/secteur) du catalogue suivant (ici un seul secteur de catalogue est en service, il n'y a donc pas de suivant)
C302-	octets n°02	n° de l'octet de la première entrée libre (#00 si plein)
C303-	octets n°03/0F	contient toujours #00 (inutilisés)
C310-	octets n°10/FF	15 "entrées" de catalogue (une ligne de 16 octets par entrée)

<u>Chaque "entrée" de catalogue est structurée ainsi:</u>

Octets n°00 à 08	nom complété à droite par des espaces (#20)

octets n°09 à 0B	extension (idem)
octet n°0C	piste du descripteur
octet n°0D	secteur du descripteur
octet n°0E	nombre de secteurs du fichier (y compris le(s) descripteurs)
octet n°0F	attribut de protection (b6=1, PROT si b7=1, UNPROT si b7=0) (#40 = 0100 0000 pour UNPROT et #C0 = 1100 0000 pour PROT). b0 à b5 = HH du nombre de secteurs = rarement utilisés, sauf pour les très gros fichiers “mergés” (comme ci-dessus F.DOC).

<u>Exemples de descripteurs:</u>

Voici par exemple le début du descripteur d'un des fichiers système, celui de la BANQUE n°7:

```
0000-  00 00 FF 40 00 C4 FF C7 00 00 04 00 05 0B 05 0C  ...@.D.G........
0010-  05 0D 05 0E 00 00 00 00 00 00 00 00 00 00 00 00  ................ etc
```

<u>Le premier secteur de descripteur</u> chargé dans BUF1 est structuré ainsi:

C100-	octets n°00/01	"lien" (coordonnées du descripteur suivant). Ici le #0000 pointe sur le secteur n°0 de la piste n°0, ce qui indique qu'il n'y a pas d'autre descripteur, car un numéro de secteur ne peut jamais être nul.
C102-	octet n°02	contient #FF (seulement si premier secteur descripteur) (Le pointeur X est positionné sur ce #FF, et permet de lire la suite)
C103-	octet n°03	(C101+X) type de fichier (voir manuel page 100). Ici #40, soit 0100 0000, indique qu'il s'agit d'un fichier de type "Bloc de données" (b6 à 1).
C104-	octets n°04/05	(C102+X et C103+X) adresse (normale) de début (ou nombre de fiches pour un fichier à accès direct). Ici #C400 est le début de la BANQUE n°7 en RAM overlay.
C106-	octets n°06/07	(C104+X et C105+X) adresse (normale) de fin (ou longueur d'une fiche pour un fichier à accès direct). Ici #C7FF est la fin de la BANQUE n°7 en RAM overlay.
C108-	octets n°08/09	(C106+X et C107+X) adresse d'exécution si AUTO, #0000 si non AUTO.
C10A-	octets n°0A/0B	(C108+X et C109+X) nombre de secteurs à charger. La BANQUE n°7 comporte 4 secteurs, d'où le #0004 qui figure ici.
C10C-	octets n°0C/FF	(C100+Y et C101+Y) liste coordonnées piste/secteur des secteurs à charger. Ici le premier secteur de la BANQUE n°7 se trouve au secteur n°11 (#0B) de la piste n°5 (#05), le dernier au secteur n°14 (#0E) de cette même piste. Dans un premier descripteur il y a de la place pour 122 paires de 2 octets. Si le nombre de secteurs à charger dépasse 122, lorsque Y atteint #00 (fin BUF1), il faut charger <u>le descripteur suivant</u> dont la structure est simplifiée:
C100-	octets n°00/01	"lien" (coordonnées du descripteur suivant)
C102-	octets n°02/FF	(C100+Y et C101+Y) liste des coordonnées piste/secteur des secteurs à charger (Y de #02 à #EF au maximum), 127 paires de 2 octets. Si le nombre de secteurs à charger dépasse 122 + 127 = 249, il faut charger le descripteur suivant etc...

Examinons maintenant les descripteurs de chacun des fichiers qui nous intéressent:

Petit Fichier A (1 seul descripteur):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  00 00 FF 40 00 10 FF 11 00 00 02 00 05 0B 05 0C  ................
C110-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

Petit Fichier B (1 seul descripteur):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  00 00 FF 40 00 20 FF 22 00 00 03 00 05 0E 05 0F  ................
C110-  05 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

Petit Fichier C (1 seul descripteur):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  00 00 FF 40 00 30 FF 30 00 00 01 00 06 01 00 00  ................
C110-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

Gros Fichier D (3 descripteurs) premier descripteur (les 2 premiers octets indiquent les coordonnées du descripteur suivant, soit le neuvième secteur de la piste #19):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  15 09 FF 40 00 04 FF FF 00 00 FC 00 0D 0F 0D 10  ................
C110-  0D 11 0E 01 0E 02 0E 03 0E 04 0E 05 0E 06 0E 07  ................
C120-  0E 08 0E 09 0E 0A 0E 0B 0E 0C 0E 0D 0E 0E 0E 0F  ................
C130-  0E 10 0E 11 0F 01 0F 02 0F 03 0F 04 0F 05 0F 06  ................
C140-  0F 07 0F 08 0F 09 0F 0A 0F 0B 0F 0C 0F 0D 0F 0E  ................
C150-  0F 0F 0F 10 0F 11 10 01 10 02 10 03 10 04 10 05  ................
C160-  10 06 10 07 10 08 10 09 10 0A 10 0B 10 0C 10 0D  ................
C170-  10 0E 10 0F 10 10 10 11 11 01 11 02 11 03 11 04  ................
C180-  11 05 11 06 11 07 11 08 11 09 11 0A 11 0B 11 0C  ................
C190-  11 0D 11 0E 11 0F 11 10 11 11 12 01 12 02 12 03  ................
C1A0-  12 04 12 05 12 06 12 07 12 08 12 09 12 0A 12 0B  ................
C1B0-  12 0C 12 0D 12 0E 12 0F 12 10 12 11 13 01 13 02  ................
C1C0-  13 03 13 04 13 05 13 06 13 07 13 08 13 09 13 0A  ................
C1D0-  13 0B 13 0C 13 0D 13 0E 13 0F 13 10 13 11 14 05  ................
C1E0-  14 06 14 08 14 09 14 0B 14 0C 14 0E 14 0F 14 11  ................
C1F0-  15 01 15 02 15 03 15 04 15 05 15 06 15 07 15 08  ................
```

Deuxième descripteur (situé dans le secteur #09 de la piste #15, les 2 premiers octets indiquent les coordonnées du descripteur suivant, soit premier secteur de la piste #1D):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  1D 01 15 0A 15 0B 15 0C 15 0D 15 0E 15 0F 15 10  ................
C110-  15 11 16 01 16 02 16 03 16 04 16 05 16 06 16 07  ................
C120-  16 08 16 09 16 0A 16 0B 16 0C 16 0D 16 0E 16 0F  ................
C130-  16 10 16 11 17 01 17 02 17 03 17 04 17 05 17 06  ................
C140-  17 07 17 08 17 09 17 0A 17 0B 17 0C 17 0D 17 0E  ................
C150-  17 0F 17 10 17 11 18 01 18 02 18 03 18 04 18 05  ................
C160-  18 06 18 07 18 08 18 09 18 0A 18 0B 18 0C 18 0D  ................
C170-  18 0E 18 0F 18 10 18 11 19 01 19 02 19 03 19 04  ................
C180-  19 05 19 06 19 07 19 08 19 09 19 0A 19 0B 19 0C  ................
C190-  19 0D 19 0E 19 0F 19 10 19 11 1A 01 1A 02 1A 03  ................
C1A0-  1A 04 1A 05 1A 06 1A 07 1A 08 1A 09 1A 0A 1A 0B  ................
C1B0-  1A 0C 1A 0D 1A 0E 1A 0F 1A 10 1A 11 1B 01 1B 02  ................
C1C0-  1B 03 1B 04 1B 05 1B 06 1B 07 1B 08 1B 09 1B 0A  ................
C1D0-  1B 0B 1B 0C 1B 0D 1B 0E 1B 0F 1B 10 1B 11 1C 01  ................
C1E0-  1C 02 1C 03 1C 04 1C 05 1C 06 1C 07 1C 08 1C 09  ................
C1F0-  1C 0A 1C 0B 1C 0C 1C 0D 1C 0E 1C 0F 1C 10 1C 11  ................
```

Troisième et dernier descripteur (situé dans le secteur #01 de la piste #1D, les 2 premiers octets indiquent #0000, c'est à dire qu'il n'y a pas de descripteur suivant):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  00 00 1D 02 1D 03 1D 04 00 00 00 00 00 00 00 00  ................
C110 à C1FF idem uniquement des zéros...
```

<u>Gros Fichier E (3 descripteurs)</u> premier descripteur (les 2 premiers octets indiquent les coordonnées du descripteur suivant, soit le secteur n°#0E de la piste #03 de la deuxième face):

```
C100-  83 0E FF 40 00 00 FF FF 00 00 00 01 26 0B 26 0C  ................
C110-  26 0D 26 0E 26 0F 26 10 26 11 27 01 27 02 27 03  ................
C120-  27 04 27 05 27 06 27 07 27 08 27 09 27 0A 27 0B  ................
C130-  27 0C 27 0D 27 0E 27 0F 27 10 27 11 28 01 28 02  ................
C140-  28 03 28 04 28 05 28 06 28 07 28 08 28 09 28 0A  ................
C150-  28 0B 28 0C 28 0D 28 0E 28 0F 28 10 28 11 29 01  ................
C160-  29 02 29 03 29 04 29 05 29 06 29 07 29 08 29 09  ................
C170-  29 0A 29 0B 29 0C 29 0D 29 0E 29 0F 29 10 29 11  ................
C180-  80 01 80 02 80 03 80 04 80 05 80 06 80 07 80 08  ................
C190-  80 09 80 0A 80 0B 80 0C 80 0D 80 0E 80 0F 80 10  ................
C1A0-  80 11 81 01 81 02 81 03 81 04 81 05 81 06 81 07  ................
C1B0-  81 08 81 09 81 0A 81 0B 81 0C 81 0D 81 0E 81 0F  ................
C1C0-  81 10 81 11 82 01 82 02 82 03 82 04 82 05 82 06  ................
C1D0-  82 07 82 08 82 09 82 0A 82 0B 82 0C 82 0D 82 0E  ................
C1E0-  82 0F 82 10 82 11 83 01 83 02 83 03 83 04 83 05  ................
C1F0-  83 06 83 07 83 08 83 09 83 0A 83 0B 83 0C 83 0D  ................
```

Deuxième descripteur (situé dans le secteur #0E de la piste #03 de la deuxième face, les premiers octets indiquent que le descripteur suivant se trouve au sixième secteur de la piste #0B de la deuxième face):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  8B 06 83 0F 83 10 83 11 84 01 84 02 84 03 84 04  ................
C110-  84 05 84 06 84 07 84 08 84 09 84 0A 84 0B 84 0C  ................
C120-  84 0D 84 0E 84 0F 84 10 84 11 85 01 85 02 85 03  ................
C130-  85 04 85 05 85 06 85 07 85 08 85 09 85 0A 85 0B  ................
C140-  85 0C 85 0D 85 0E 85 0F 85 10 85 11 86 01 86 02  ................
C150-  86 03 86 04 86 05 86 06 86 07 86 08 86 09 86 0A  ................
C160-  86 0B 86 0C 86 0D 86 0E 86 0F 86 10 86 11 87 01  ................
C170-  87 02 87 03 87 04 87 05 87 06 87 07 87 08 87 09  ................
C180-  87 0A 87 0B 87 0C 87 0D 87 0E 87 0F 87 10 87 11  ................
C190-  88 01 88 02 88 03 88 04 88 05 88 06 88 07 88 08  ................
C1A0-  88 09 88 0A 88 0B 88 0C 88 0D 88 0E 88 0F 88 10  ................
C1B0-  88 11 89 01 89 02 89 03 89 04 89 05 89 06 89 07  ................
C1C0-  89 08 89 09 89 0A 89 0B 89 0C 89 0D 89 0E 89 0F  ................
C1D0-  89 10 89 11 8A 01 8A 02 8A 03 8A 04 8A 05 8A 06  ................
C1E0-  8A 07 8A 08 8A 09 8A 0A 8A 0B 8A 0C 8A 0D 8A 0E  ................
C1F0-  8A 0F 8A 10 8A 11 8B 01 8B 02 8B 03 8B 04 8B 05  ................
```

Troisième et dernier descripteur (situé dans le sixième secteur de la piste #0B de la deuxième face, les 2 premiers octets indiquent #0000, c'est à dire qu'il n'y a pas de descripteur suivant):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  00 00 8B 07 8B 08 8B 09 8B 0A 8B 0B 8B 0C 8B 0D  ................
C110-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

<u>Descripteurs des fichiers “mergés”</u>

Pour les fichiers “mergés”

Le "lien" du dernier descripteur de chaque fichier indique les coordonnées du premier descripteur du fichier suivant (le "lien" du dernier descripteur du dernier fichier indique bien sûr #0000). Il semble possible de combiner les descripteurs pour gagner de la place. Dans ce cas, un #FF sera placé après la dernière paire de coordonnées piste/secteur du dernier secteur à charger à la fin du dernier descripteur de chaque fichier et sera suivi des octets usuels: type de fichier, adresse (normale) de début, adresse (normale) de fin, adresse d'exécution, nombre de secteurs à charger, liste des coordonnées piste/secteur des secteurs à charger du fichier “mergé”. La présence du #FF valide la valeur de X pour la lecture des octets de STATUS, puis la valeur de Y pour la lecture des octets de coordonnées piste/secteur des secteurs à charger. Le premier descripteur de chaque fichier, dont les coordonnées figurent dans “l’entrée” du catalogue, est le descripteur "principal".

Exemples de descripteurs de fichiers “mergés”

Petit Fichier G (3 descripteurs) (formé des 3 petits fichiers A, B et C “mergés”)
premier descripteur (fichier A), avec les coordonnées du descripteur du fichier B):

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  06 05 FF 40 00 10 FF 11 00 00 02 00 06 03 06 04  ................
C110-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

Deuxième descripteur (fichier B), avec les coordonnées du descripteur du fichier C):

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  06 09 FF 40 00 20 FF 22 00 00 03 00 06 06 06 07  ................
C110-  06 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

Troisième et dernier descripteur (fichier C):

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  00 00 FF 40 00 30 FF 30 00 00 01 00 06 0A 00 00  ................
C110-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

Gros Fichier F (6 descripteurs) (formé des gros fichiers D et E “mergés”)
premier descripteur (fichier D), avec les coordonnées du deuxième descripteur de D):

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  93 01 FF 40 00 00 FF FF 00 00 00 01 8B 0F 8B 10  ................
C110-  8B 11 8C 01 8C 02 8C 03 8C 04 8C 05 8C 06 8C 07  ................
C120-  8C 08 8C 09 8C 0A 8C 0B 8C 0C 8C 0D 8C 0E 8C 0F  ................
C130-  8C 10 8C 11 8D 01 8D 02 8D 03 8D 04 8D 05 8D 06  ................
C140-  8D 07 8D 08 8D 09 8D 0A 8D 0B 8D 0C 8D 0D 8D 0E  ................
C150-  8D 0F 8D 10 8D 11 8E 01 8E 02 8E 03 8E 04 8E 05  ................
C160-  8E 06 8E 07 8E 08 8E 09 8E 0A 8E 0B 8E 0C 8E 0D  ................
C170-  8E 0E 8E 0F 8E 10 8E 11 8F 01 8F 02 8F 03 8F 04  ................
```

```
C180-  8F 05 8F 06 8F 07 8F 08 8F 09 8F 0A 8F 0B 8F 0C  ................
C190-  8F 0D 8F 0E 8F 0F 8F 10 8F 11 90 01 90 02 90 03  ................
C1A0-  90 04 90 05 90 06 90 07 90 08 90 09 90 0A 90 0B  ................
C1B0-  90 0C 90 0D 90 0E 90 0F 90 10 90 11 91 01 91 02  ................
C1C0-  91 03 91 04 91 05 91 06 91 07 91 08 91 09 91 0A  ................
C1D0-  91 0B 91 0C 91 0D 91 0E 91 0F 91 10 91 11 92 01  ................
C1E0-  92 02 92 03 92 04 92 05 92 06 92 07 92 08 92 09  ................
C1F0-  92 0A 92 0B 92 0C 92 0D 92 0E 92 0F 92 10 92 11  ................
```

Deuxième descripteur (fichier D), avec les coordonnées du troisième descripteur de D):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  9A 0A 93 02 93 03 93 04 93 05 93 06 93 07 93 08  ................
C110-  93 09 93 0A 93 0B 93 0C 93 0D 93 0E 93 0F 93 10  ................
C120-  93 11 94 01 94 02 94 03 94 04 94 05 94 06 94 07  ................
C130-  94 08 94 09 94 0A 94 0B 94 0C 94 0D 94 0E 94 0F  ................
C140-  94 10 94 11 95 01 95 02 95 03 95 04 95 05 95 06  ................
C150-  95 07 95 08 95 09 95 0A 95 0B 95 0C 95 0D 95 0E  ................
C160-  95 0F 95 10 95 11 96 01 96 02 96 03 96 04 96 05  ................
C170-  96 06 96 07 96 08 96 09 96 0A 96 0B 96 0C 96 0D  ................
C180-  96 0E 96 0F 96 10 96 11 97 01 97 02 97 03 97 04  ................
C190-  97 05 97 06 97 07 97 08 97 09 97 0A 97 0B 97 0C  ................
C1A0-  97 0D 97 0E 97 0F 97 10 97 11 98 01 98 02 98 03  ................
C1B0-  98 04 98 05 98 06 98 07 98 08 98 09 98 0A 98 0B  ................
C1C0-  98 0C 98 0D 98 0E 98 0F 98 10 98 11 99 01 99 02  ................
C1D0-  99 03 99 04 99 05 99 06 99 07 99 08 99 09 99 0A  ................
C1E0-  99 0B 99 0C 99 0D 99 0E 99 0F 99 10 99 11 9A 01  ................
C1F0-  9A 02 9A 03 9A 04 9A 05 9A 06 9A 07 9A 08 9A 09  ................
```

Troisième descripteur (fichier D), avec les coordonnées du premier du fichier E):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  9B 01 9A 0B 9A 0C 9A 0D 9A 0E 9A 0F 9A 10 9A 11  ................
C110-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

Quatrième descripteur (fichier E), avec les coordonnées du deuxième descripteur de E):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  A2 05 FF 40 00 04 FF FF 00 00 FC 00 9B 02 9B 03  ................
C110-  9B 04 9B 05 9B 06 9B 07 9B 08 9B 09 9B 0A 9B 0B  ................
C120-  9B 0C 9B 0D 9B 0E 9B 0F 9B 10 9B 11 9C 01 9C 02  ................
C130-  9C 03 9C 04 9C 05 9C 06 9C 07 9C 08 9C 09 9C 0A  ................
C140-  9C 0B 9C 0C 9C 0D 9C 0E 9C 0F 9C 10 9C 11 9D 01  ................
C150-  9D 02 9D 03 9D 04 9D 05 9D 06 9D 07 9D 08 9D 09  ................
C160-  9D 0A 9D 0B 9D 0C 9D 0D 9D 0E 9D 0F 9D 10 9D 11  ................
C170-  9E 01 9E 02 9E 03 9E 04 9E 05 9E 06 9E 07 9E 08  ................
C180-  9E 09 9E 0A 9E 0B 9E 0C 9E 0D 9E 0E 9E 0F 9E 10  ................
C190-  9E 11 9F 01 9F 02 9F 03 9F 04 9F 05 9F 06 9F 07  ................
C1A0-  9F 08 9F 09 9F 0A 9F 0B 9F 0C 9F 0D 9F 0E 9F 0F  ................
C1B0-  9F 10 9F 11 A0 01 A0 02 A0 03 A0 04 A0 05 A0 06  ................
C1C0-  A0 07 A0 08 A0 09 A0 0A A0 0B A0 0C A0 0D A0 0E  ................
C1D0-  A0 0F A0 10 A0 11 A1 01 A1 02 A1 03 A1 04 A1 05  ................
C1E0-  A1 06 A1 07 A1 08 A1 09 A1 0A A1 0B A1 0C A1 0D  ................
C1F0-  A1 0E A1 0F A1 10 A1 11 A2 01 A2 02 A2 03 A2 04  ................
```

Cinquième descripteur (fichier E), avec les coordonnées du troisième descripteur de E):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  A9 0E A2 06 A2 07 A2 08 A2 09 A2 0A A2 0B A2 0C  ................
C110-  A2 0D A2 0E A2 0F A2 10 A2 11 A3 01 A3 02 A3 03  ................
C120-  A3 04 A3 05 A3 06 A3 07 A3 08 A3 09 A3 0A A3 0B  ................
C130-  A3 0C A3 0D A3 0E A3 0F A3 10 A3 11 A4 01 A4 02  ................
C140-  A4 03 A4 04 A4 05 A4 06 A4 07 A4 08 A4 09 A4 0A  ................
C150-  A4 0B A4 0C A4 0D A4 0E A4 0F A4 10 A4 11 A5 01  ................
C160-  A5 02 A5 03 A5 04 A5 05 A5 06 A5 07 A5 08 A5 09  ................
C170-  A5 0A A5 0B A5 0C A5 0D A5 0E A5 0F A5 10 A5 11  ................
C180-  A6 01 A6 02 A6 03 A6 04 A6 05 A6 06 A6 07 A6 08  ................
C190-  A6 09 A6 0A A6 0B A6 0C A6 0D A6 0E A6 0F A6 10  ................
C1A0-  A6 11 A7 01 A7 02 A7 03 A7 04 A7 05 A7 06 A7 07  ................
C1B0-  A7 08 A7 09 A7 0A A7 0B A7 0C A7 0D A7 0E A7 0F  ................
C1C0-  A7 10 A7 11 A8 01 A8 02 A8 03 A8 04 A8 05 A8 06  ................
C1D0-  A8 07 A8 08 A8 09 A8 0A A8 0B A8 0C A8 0D A8 0E  ................
C1E0-  A8 0F A8 10 A8 11 A9 01 A9 02 A9 03 A9 04 A9 05  ................
C1F0-  A9 06 A9 07 A9 08 A9 09 A9 0A A9 0B A9 0C A9 0D  ................
```

Sixième et dernier descripteur (fichier N, c'est le troisième du fichier E):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100-  00 00 A9 0F A9 10 A9 11 00 00 00 00 00 00 00 00  ................
C110-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

ANNEXE n° 8
Que se passe t-il lors d’un SAVE?

Nombreux sont ceux qui veulent savoir comment SEDORIC sauve un fichier, afin de pouvoir implémenter cette fonction essentielle dans un programme en cours de développement. Il est bien sûr possible d'aller fouiner dans les dédales de la commande SAVE. Mais c'est ardu et un résumé des modifications subies par la disquette lors d'un SAVE m'a souvent été demandé.

Voici donc, **en très simplifié**, ce que fait SEDORIC, sans entrer dans les détails tels que gestion des paramètres: nom de fichier, type de fichier, adresses, vérification des noms de fichiers déjà existants, mise à jour de diverses variables, etc. :

1) SEDORIC cherche dans la bitmap (BUF2) le premier secteur libre et le réserve (le marque occupé) pour y mettre le premier descripteur. Il élabore le premier descripteur (BUF1), ce faisant il cherche les secteurs libres suivants et les réserve (les marque occupés) pour les secteurs de data proprement dits. Il copie le premier descripteur (BUF1) dans le secteur réservé à cet effet. Si nécessaire, cherche un nouveau secteur libre, élabore un deuxième descripteur etc...

2) SEDORIC écrit tous les secteurs de data en se basant sur la liste des secteurs réservés précédemment dans le ou les descripteurs.

3) Enfin il cherche la première entrée libre dans les secteurs de directory (BUF3). Sauve la bitmap (BUF2). Ecrit la nouvelle "entrée" (nom du fichier etc) dans le secteur de directory (BUF3) et sauve celui-ci.

En pratique, peu d'octets sont modifiés sur la disquette: outre évidemment les secteurs de data proprement dits et les descripteurs correspondant qui ont été écrits, une nouvelle "entrée" (16 octets) a été ajoutée dans le directory et la bitmap a été mise à jour. C'est tout!

Oublions les procédures un peu complexes qui sont utilisées par SEDORIC et voyons maintenant les quelques modifications qu'un SAVE apporte sur la disquette. Je partirai d'un exemple simple, pour ceux qui voudraient aller voir eux-mêmes ce qui se passe. Soit une disquette vierge Master formatée avec un INIT A,16,80,D.

J'ai choisi cette configuration pour plusieurs raisons: double face afin de voir ce qui se passe quand on écrit un fichier "à cheval" sur les deux faces, 80 pistes afin de voir ce qui se passe quand le deuxième secteur de bitmap est utilisé et enfin 16 secteurs par piste afin de simplifier la lecture de la bitmap. En effet, dans ce dernier cas, chaque secteur étant représenté par un bit, pour représenter toute une piste il faudra 16 bits soit 2 octets tous ronds. Il est alors possible de déchiffrer la bitmap "à l'oeil nu". Après ce formatage, un DIR révèle qu'il y a 2453 secteurs de libres sur cette disquette.

Si l'on boote de frais avec cette disquette et que l'on sauve un fichier (fictif) avec un SAVE"FICHIER01",A#1000,E#1EFF, la commande DIR indique alors 2437 secteurs libres et un fichier de 16 secteurs. La longueur du fichier est tout juste de 15 secteurs de data plus 1 secteur de descripteur = 16 secteurs.

Analyse des changements intervenus dans la disquette.

1) DESCRIPTEUR: SEDORIC a été à la recherche dans la bit map du premier secteur libre de la disquette, il l'a trouvé au quatrième secteur de la piste numéro 6 et y a écrit le descripteur du nouveau fichier, conformément à la structure suivante:

```
Piste 6 Secteur 4 (avant)                      Piste 6 Secteur 4 (après)
00+00000000 00000000 00000000 00000000         00+0000FF40 0010FF1E 00000F00 06050606
10+00000000 00000000 00000000 00000000         10+06070608 0609060A 060B060C 060D060E
20+00000000 00000000 00000000 0000etc.         20+060F0610 07010702 07030000 0000etc.
```

Octets 00 et 01: Lien avec le descripteur suivant (ici #**00** #**00**, ce qui indique qu'il n'y en a pas car un secteur ne peut jamais avoir le numéro zéro). Ce lien n'est utilisé que pour les gros fichiers (plus de 122 secteurs de data). Lorsqu'il existe, il est constitué du numéro de piste puis du numéro de secteur où se trouve le descripteur suivant.

Octet 02: Ici, toujours #**FF** pour un premier descripteur.

Octet 03: Type de fichier (ici #**40**, bloc de data). Rappel du codage de cet octet: b0 à 1 si AUTO, b1 et b2 non utilisés, b3 à 1 si fichier direct, b4 à 1 si fichier séquentiel, b5 et b6 à 1 si fichier window, b6 à 1 si bloc de data et b7 à 1 si BASIC.

Octets 04 et 05: Adresse de début du fichier (ici #**00** et #**10** pour #1000).

Octets 06 et 07: Adresse de fin du fichier, ou longueur d'une fiche pour un fichier à accès direct (ici #**FF** et #**1E** pour #1EFF).

Octets 08 et 09: Adresse d'exécution si AUTO (sinon #**00** et #**00**,comme dans notre exemple).

Octets 0A et 0B: Nombre de secteurs de data (ici #**0F** et #**00** pour #000F soit 15). Notons que le dernier secteur de data est la plupart du temps incomplet, mais ici la longueur de notre fichier est un multiple de 256 octet, donc le dernier secteur de data est complet.

Octets 0C à FF: liste des coordonnées piste/secteur de chacun de ces secteurs de data (pour 122 secteurs au maximum dans un descripteur de ce type). Ici on commence avec #**06** et #**05**, ce qui indique que le premier secteur de data a été écrit dans le cinquième secteur de la sixième piste, c'est à dire dans le secteur libre suivant, juste après le descripteur. La quinzième et dernière paire de coordonnées est #**07** et #**03**. Le dernier secteur de data a donc été écrit dans le troisième secteur de la septième piste.

Lorsqu'il n'y a pas assez de place pour écrire les coordonnées piste/secteur de tous les secteurs de data dans ce descripteur, SEDORIC élabore un "descripteur suivant" dont la structure est simplifiée:

Octets 00 et 01: Lien avec le descripteur suivant, s'il existe.

Octets 02 à FF: liste des coordonnées piste/secteur (pour 127 secteurs au maximum dans un descripteur de ce type).

Pour les fichiers "mergés", formés en fait de plusieurs fichiers à la queue leu-leu, le lien du dernier descripteur de chaque fichier indique les coordonnées du premier descripteur du fichier suivant. Le premier

descripteur du premier fichier est alors appelé "descripteur principal" et ce sont ses coordonnées qui seront indiquées dans "l'entrée" du directory.

2) BITMAP: SEDORIC a ensuite indiqué dans le premier secteur de bitmap que ce fichier a été écrit. Seuls les octets suivants ont été modifiés:

```
Piste 20 Secteur 2 (avant)
00+FF009509 00005010 01D00000 00000000
10+00000000 00000000 00000000 F8FFFFFF
20+FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
30+FFFFFFFF FFFFFFFF B06DFFFF FFFFetc.
```

```
Piste 20 Secteur 2 (après)
00+FF008509 01005010 01D00000 00000000
10+00000000 00000000 00000000 0000F8FF
20+FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
30+FFFFFFFF FFFFFFFF B06DFFFF FFFFetc.
```

Octets 02 et 03: Nombre de secteurs libres, ici #**85** et #**09** pour #0985 (2437) au lieu de #95 et #09 pour #0995 (2453).

Octets 04 et 05: Nombre de fichiers en tout (ici #**01** et #**00** soit #0001).

Octet 08: Nombre de secteurs de directory. Ici on avait #01 car même pour une disquette vierge, le premier secteur de directory est validé. Cette valeur est restée inchangée. Mais ce n'est évidemment pas toujours le cas. En effet, lorsque le nombre de fichiers dépasse 15, SEDORIC valide un second secteur de directory, situé au septième secteur de la piste #14 (20). Et ainsi de suite. Les secteurs numéros #04, #07, #0A, #0D et #10 de la piste #14 sont en effet déjà réservés, c'est à dire marqués occupés, même sur une disquette vierge. Si le nombre de fichiers dépasse la capacité de ces 5 secteurs réservés, c'est à dire 15x5=75 fichiers, SEDORIC écrit un nouveau secteur de directory dans le premier secteur libre qu'il trouve et cette fois il bascule le bit correspondant dans la bitmap. Et ainsi de suite.

Octets 10 à FF: Bitmap proprement dite. La disquette est considérée comme une suite de secteurs (80x16x2=2560 secteurs dans notre cas). Chaque secteur est représenté par un bit. Ces bits sont rangés par groupes de 8, dans des octets (8 bits) Dans chaque octet, ils sont placés de b0 à b7, il faut donc les lire "de droite à gauche". Ces octets sont rangés dans l'ordre croissant à partir de l'octet numéro #10, il faut donc les lire de "gauche à droite". Dans notre exemple, une piste (16 secteurs) est codée sur deux octets. La première moitié de la piste zéro, par exemple, est codée en #10, la seconde moitié en #11. Puis la première moitié de la piste 1 etc. Lorsqu'un octet est libre, le bit correspondant est à 1. SEDORIC bascule ce bit à zéro pour indiquer que le secteur est occupé.

Dans notre exemple, le descripteur a été écrit au quatrième secteur de la piste numéro 6, puis les 15 secteurs de data ont été écrits à la suite, jusqu'au troisième secteur de la septième piste. La piste numéro 6 (septième piste) est représentée par les octets #1C et #1D qui indiquaient #F8 et #FF et qui indiquent maintenant #**00** et #**00** soit 1111 1000 1111 1111 avant et 0000 0000 0000 0000 après. Attention, les 3 zéros soulignés indiquent que les 3 premiers secteurs de la piste numéro 6 étaient occupés et que le reste de cette piste était libre. Après écriture du fichier, tout la piste numéro 6 est occupée. Mais le fichier va plus loin, puisqu'il se termine sur la piste numéro 7. Sa fin est donc codée dans les octets numéros #1E et #1F. Avant écriture, ces octets valaient #FF et #FF et après #**F8** et #**FF** soit 1111 1111 1111 1111 avant et 1111 1000 1111 1111 après. La piste 7 était complètement libre. Maintenant, les 3 zéros soulignés indique que les 3 premiers secteurs de la piste numéro 7 sont occupés, le reste de cette piste restant toujours libre.

Ce premier secteur de bitmap permet de coder 1919 secteurs. Lorsque ce chiffre est dépassé, il peut encore coder 1919 secteurs sur le deuxième secteur de bitmap (troisième secteur de la piste #14). Ce deuxième

secteur de bitmap est codé exactement comme le premier, mais il n'est mis à jour que lorsqu'il est utilisé (voir plus loin).

Piste 20 Secteur 3 (2e Secteur de Bitmap)	Piste 20 Secteur 3 (2e Sec. de Bitmap)
00+FF00000A 00005010 01D00000 00000000	00+FF00000A 00005010 01D00000 00000000
10+FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	10+FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
20+FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	20+FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
30+FFFFFFFF FFFFFFFF FFFFFFFF FFFFetc.	30+FFFFFFFF FFFFFFFF FFFFFFFF FFFFetc.

3) DIRECTORY: Enfin SEDORIC ajoute une entrée dans le directory. Les octets suivants sont modifiés:

Piste 20 Secteur 4 (1er Sec. Directory)	Piste 20 Secteur 4 (1er Sec Directory)
00+00001000 00000000 00000000 00000000	00+0000**20**00 00000000 00000000 00000000
10+00000000 00000000 00000000 00000000	10+**46494348 49455230 31434F4D 06041040**
	F I C H I E R 0 1 C O M
20+00000000 00000000 00000000 0000etc.	20+00000000 00000000 00000000 0000etc.

Octet 02: n° de l'octet de la première "entrée" libre. Ici la valeur #10 est changée en #**20**. Les caractéristiques du nouveau fichier sont portées sur la ligne #10 à #1F et le prochain fichier sera décrit à partir de #20.

Octets #X0 à #XF: "Entrée" pour le nouveau fichier. Dans notre exemple, de #10 à #1F, écriture des octets suivants:

Octets #X0 à #X8: nom du fichier, complété à droite par des espaces (#20) si nécessaire. Dans notre exemple, de #10 à #18, on a #**46**, #**49**, #**43**, #**48**, #**49**, #**45**, #**52**, #**30** et #**31** soit "FICHIER01"

Octets #X9 à #B: extension, complétée à droite par des espaces (#20) si nécessaire. Dans notre exemple, de #19 à #1B, on a #**43**, #**4F** et #**4D** soit "COM"

Octet #XC: piste du descripteur (ou du premier descripteur pour les gros fichiers ou les fichiers mergés). Dans notre exemple, en #1C, on a #**06** pour piste numéro 6.

Octet #XD: secteur du descripteur (ou du premier descripteur pour les gros fichiers ou les fichiers mergés). Dans notre exemple, en #1D, on a #**04** pour secteur numéro 4.

Octet #XE: LL (octet de poids faible) du nombre total de secteurs du fichier. Dans notre exemple, en #1E, on a #**10** car la taille totale du fichier est de 16 secteurs.

Octet #XF: HH (octet de poids fort) du nombre total de secteurs du fichier (de b0 à b5) et attibut de protection (b6 toujours à 1 et b7 à 0 si UNPROT ou à 1 si PROT). Dans notre exemple, en #1F, on a #**40**: seul le b6 est à 1, car la taille de notre fichier est codée seulement sur l'octet de poids faible et il n'est pas protégé.

Que se passe t-il quand un fichier "déborde" sur la deuxième face de la disquette?

Au fur et à mesure que SEDORIC écrit des fichiers sur la disquettes, celle-ci subit des modifications analogues à celles qui ont été décrites ci-dessus: écriture d'un ou de plusieurs descripteurs, écriture des secteurs de data, mise à jour de la bitmap et mise à jour du directory.

Rappelons que dans tous les cas où SEDORIC doit utiliser une nouvelle place, il recherche de façon séquentielle la première qui est libre. Ceci vaut aussi bien pour un nouveau secteur (descripteur, data et même directory, lorsque les secteurs de directory réservés sont pleins) que pour une nouvelle entrée dans les secteurs de directory.

Dans une disquette où certains fichiers ont été effacés, il reste des "trous" que SEDORIC utilise donc en priorité. Voici une illustration du passage à la deuxième face:

```
Directory (avant)
Drive A V3 (Mst)             XX/XX/XX
FICHIER01.COM  16   FICHIER02.COM  16
FICHIER03.COM 100   FICHIER04.COM 100
FICHIER05.COM 100   FICHIER06.COM 100
FICHIER07.COM 100   FICHIER08.COM 100
FICHIER09.COM 100   FICHIER10.COM 100
FICHIER11.COM 100   FICHIER12.COM 100
FICHIER13.COM 100
1321 sectors free (D/80/16) 13 Files

Piste 20 Secteur 2 (avant)
00+FF002905 0D005010 01D00000 .../...
A0+00000000 00000000 000080FF FFFFFFFF
B0+FFFFFFFF FFFFFFFF FFFFFFFF FFFFetc.

Piste 20 Secteur 3 (avant)
00+FF00000A 00005010 01D00000 00000000
10+FFFFFFFF FFFFFFFF FFFFFFFF FFFFetc.

Piste 20 Secteur 4 (avant)
00+0000E000 00000000 00000000 00000000
10+46494348 49455230 31434F4D 06041040
   F I C H  I E R 0  1 C O M
20+46494348 49455230 32434F4D 07041040
   F I C H  I E R 0  2 C O M
30+46494348 49455230 33434F4D 08046440
   F I C H  I E R 0  3 C O M
40+46494348 49455230 34434F4D 0E086440
   F I C H  I E R 0  4 C O M
50+46494348 49455230 35434F4D 15046440
   F I C H  I E R 0  5 C O M
60+46494348 49455230 36434F4D 1B086440
   F I C H  I E R 0  6 C O M
70+46494348 49455230 37434F4D 210C6440
   F I C H  I E R 0  7 C O M
80+46494348 49455230 38434F4D 27106440
   F I C H  I E R 0  8 C O M
90+46494348 49455230 39434F4D 2E046440
   F I C H  I E R 0  9 C O M
A0+46494348 49455231 30434F4D 34086440
   F I C H  I E R 1  0 C O M
B0+46494348 49455231 31434F4D 3A0C6440
   F I C H  I E R 1  1 C O M
C0+46494348 49455231 32434F4D 40106440
   F I C H  I E R 1  2 C O M
D0+46494348 49455231 33434F4D 47046440
   F I C H  I E R 1  3 C O M
E0+00000000 00000000 00000000 00000000

F0+00000000 00000000 00000000 00000000
```

```
Directory (après)
Drive A V3 (Mst)             XX/XX/XX
FICHIER01.COM  16   FICHIER02.COM  16
FICHIER03.COM 100   FICHIER04.COM 100
FICHIER05.COM 100   FICHIER06.COM 100
FICHIER07.COM 100   FICHIER08.COM 100
FICHIER09.COM 100   FICHIER10.COM 100
FICHIER11.COM 100   FICHIER12.COM 100
FICHIER13.COM 100   FICHIER14.COM 100
1221 sectors free (D/80/16) 14 Files

Piste 20 Secteur 2 (après)
00+FF00C504 0E005010 01D00000 .../...
A0+00000000 00000000 00000000 00000000
B0+00000000 000000F8 FFFFFFFF FFFFetc.

Piste 20 Secteur 3 (après)
00+FF00000A 00005010 01D00000 00000000
10+FFFFFFFF FFFFFFFF FFFFFFFF FFFFetc.

Piste 20 Secteur 4 (après)
00+0000F000 00000000 00000000 00000000
10+46494348 49455230 31434F4D 06041040
           F I C H  I E R 0  1 C O M
20+46494348 49455230 32434F4D 07041040
           F I C H  I E R 0  2 C O M
30+46494348 49455230 33434F4D 08046440
           F I C H  I E R 0  3 C O M
40+46494348 49455230 34434F4D 0E086440
           F I C H  I E R 0  4 C O M
50+46494348 49455230 35434F4D 15046440
           F I C H  I E R 0  5 C O M
60+46494348 49455230 36434F4D 1B086440
           F I C H  I E R 0  6 C O M
70+46494348 49455230 37434F4D 210C6440
           F I C H  I E R 0  7 C O M
80+46494348 49455230 38434F4D 27106440
           F I C H  I E R 0  8 C O M
90+46494348 49455230 39434F4D 2E046440
           F I C H  I E R 0  9 C O M
A0+46494348 49455231 30434F4D 34086440
           F I C H  I E R 1  0 C O M
B0+46494348 49455231 31434F4D 3A0C6440
           F I C H  I E R 1  1 C O M
C0+46494348 49455231 32434F4D 40106440
           F I C H  I E R 1  2 C O M
D0+46494348 49455231 33434F4D 47046440
           F I C H  I E R 1  3 C O M
E0+46494348 49455231 34434F4D 4D086440
           F I C H  I E R 1  4 C O M
F0+00000000 00000000 00000000 00000000
```

```
Piste 77 Secteur 8 (avant)
00+00000000 00000000 00000000 00000000
10+00000000 00000000 00000000 00000000
20+00000000 00000000 00000000 00000000
30+00000000 00000000 00000000 00000000
40+00000000 00000000 00000000 00000000
50+00000000 00000000 00000000 00000000
60+00000000 00000000 00000000 00000000
70+00000000 00000000 00000000 00000000
80+00000000 00000000 00000000 00000000
90+00000000 00000000 00000000 00000000
A0+00000000 00000000 00000000 00000000
B0+00000000 00000000 00000000 00000000
C0+00000000 00000000 00000000 00000000
D0+00000000 00000000 00000000 .../...
```

```
Piste  77  Secteur 8 (Après = nouv. desc.)
00+0000FF40 0010FF72 00006300 4D094D0A
10+4D0B4D0C 4D0D4D0E 4D0F4D10 4E014E02
20+4E034E04 4E054E06 4E074E08 4E094E0A
30+4E0B4E0C 4E0D4E0E 4E0F4E10 4F014F02
40+4F034F04 4F054F06 4F074F08 4F094F0A
50+4F0B4F0C 4F0D4F0E 4F0F4F10 80018002
60+80038004 80058006 80078008 8009800A
70+800B800C 800D800E 800F8010 81018102
80+81038104 81058106 81078108 8109810A
90+810B810C 810D810E 810F8110 82018202
A0+82038204 82058206 82078208 8209820A
B0+820B820C 820D820E 820F8210 83018302
C0+83038304 83058306 83078308 8309830A
D0+830B0000 00000000 00000000 .../...
```

Que se passe t-il quand un fichier "déborde" sur la deuxième bitmap?

Quant la première bitmap est pleine, c'est à dire quant il reste moins de 1919 secteurs de libres, SEDORIC met en service la deuxième bitmap (secteur numéro 3 de la vingtième piste). Voici une illustration du passage à la deuxième bitmap, lors de l'écriture du vingtième fichier:

```
Directory (avant)
Drive A V3 (Mst)                XX/XX/XX
FICHIER01.COM  16   FICHIER02.COM  16
FICHIER03.COM 100   FICHIER04.COM 100
FICHIER05.COM 100   FICHIER06.COM 100
FICHIER07.COM 100   FICHIER08.COM 100
FICHIER09.COM 100   FICHIER10.COM 100
FICHIER11.COM 100   FICHIER12.COM 100
FICHIER13.COM 100   FICHIER14.COM 100
FICHIER15.COM 100   FICHIER16.COM 100
FICHIER17.COM 100   FICHIER18.COM 100
FICHIER19.COM 100
*721 sectors free (D/80/16) 19 Files
```

```
Directory (après)
Drive A V3 (Mst)                XX/XX/XX
FICHIER01.COM  16   FICHIER02.COM  16
FICHIER03.COM 100   FICHIER04.COM 100
FICHIER05.COM 100   FICHIER06.COM 100
FICHIER07.COM 100   FICHIER08.COM 100
FICHIER09.COM 100   FICHIER10.COM 100
FICHIER11.COM 100   FICHIER12.COM 100
FICHIER13.COM 100   FICHIER14.COM 100
FICHIER15.COM 100   FICHIER16.COM 100
FICHIER17.COM 100   FICHIER18.COM 100
FICHIER19.COM 100   FICHIER20.COM 100
*621 sectors free (D/80/16) 20 Files
```

```
Piste 20 Secteur 2 (avant)
00+FF00D102 13005010 02D00000 .../...
F0+00000000 0080FFFF FFFFFFFF FFFFFFFF
```

```
Piste 20 Secteur 2 (après)
00+FF006D02 14005010 02D00000 .../...
F0+00000000 00000000 00000000 00000000
```

```
Piste 20 Secteur 3 (avant)
00+FF00000A 00005010 01D00000 00000000
10+FFFFFFFF FFFFFFFF FFFFFFFF FFFFetc.
```

```
Piste 20 Secteur 3 (après)
00+FF006D02 14005010 02D00000 00000000
10+0000F8FF FFFFFFFF FFFFFFFF FFFFetc.
```

```
Piste 20 Secteur 4 (avant)
00+14070000 00000000 00000000 00000000
10+46494348 49455230 31434F4D 06041040
    F I C H  I E R 0  1 C O M
.../...
F0+46494348 49455231 35434F4D 830C6440
    F I C H  I E R 1  5 C O M
```

```
Piste 20 Secteur 4 (après)
00+14070000 00000000 00000000 00000000
10+46494348 49455230 31434F4D 06041040
          F I C H  I E R 0  1 C O M
.../...
F0+46494348 49455231 35434F4D 830C6440
          F I C H  I E R 1  5 C O M
```

```
Piste 20 Secteur 7 (avant)
00+00005000 00000000 00000000 00000000
10+46494348 49455231 36434F4D 89106440
    F I C H  I E R 1  6 C O M
.../...
40+46494348 49455231 39434F4D 9C0C6440
    F I C H  I E R 1  9 C O M
50+00000000 00000000 00000000 00000000

60+00000000 00000000 00000000 0000etc.

Piste #A2 Secteur 10 (avant)
00+00000000 00000000 00000000 00000000
10+00000000 00000000 00000000 00000000
20+00000000 00000000 00000000 00000000
30+00000000 00000000 00000000 00000000
40+00000000 00000000 00000000 00000000
50+00000000 00000000 00000000 00000000
60+00000000 00000000 00000000 00000000
70+00000000 00000000 00000000 00000000
80+00000000 00000000 00000000 00000000
90+00000000 00000000 00000000 00000000
A0+00000000 00000000 00000000 00000000
B0+00000000 00000000 00000000 00000000
C0+00000000 00000000 00000000 00000000
D0+00000000 00000000 00000000 0000etc.
```

```
Piste 20 Secteur 7 (après)
00+00006000 00000000 00000000 00000000
10+46494348 49455231 36434F4D 89106440
    F I C H  I E R 1  6 C O M
.../...
40+46494348 49455231 39434F4D 9C0C6440
    F I C H  I E R 1  9 C O M
50+46494348 49455232 30434F4D A2106440
    F I C H  I E R 2  0 C O M
60+00000000 00000000 00000000 0000etc.

Piste #A2 Secteur #10 (nouveau des.)
00+0000FF40 0010FF72 00006300 A301A302
10+A303A304 A305A306 A307A308 A309A30A
20+A30BA30C A30DA30E A30FA310 A401A402
30+A403A404 A405A406 A407A408 A409A40A
40+A40BA40C A40DA40E A40FA410 A501A502
50+A503A504 A505A506 A507A508 A509A50A
60+A50BA50C A50DA50E A50FA510 A601A602
70+A603A604 A605A606 A607A608 A609A60A
80+A60BA60C A60DA60E A60FA610 A701A702
90+A703A704 A705A706 A707A708 A709A70A
A0+A70BA70C A70DA70E A70FA710 A801A802
B0+A803A804 A805A806 A807A808 A809A80A
C0+A80BA80C A80DA80E A80FA810 A901A902
D0+A9030000 00000000 00000000 0000etc.
```

ANNEXE n° 9
Que se passe t-il lors d'un DEL?

Après avoir vu quelles sont les modifications subies par la disquette lors de l'exécution de la commande SAVE, nous examinerons ce qui se passe lors d'un DEL. La commande DEL est bien plus simple que la commande SAVE. Voici un **résumé** de ce que fait SEDORIC:

1) Il analyse le NFA (nom de fichier ambigu) indiqué, cherche dans le directory le premier fichier qui correspond à ce NFA. Si le NFA comportait des "jockers", il demande s'il faut effacer ce fichier, si ce n'est pas le cas, il reprend sa recherche dans le directory.

2) Il "efface" ce fichier, restructure éventuellement le catalogue, libère les secteurs correspondant dans la bit map, décrémente le nombre de fichiers, met à jour le nombre de secteurs libres.

3) C'est fini, sauf si le NFA comportait des "jockers", auquel cas il reprend sa recherche dans le directory tant qu'il y reste des noms de fichiers à examiner.

Il y a bien sûr quelques petits rafinements supplémentaires que nous passerons sous silence. Par exemple, SEDORIC teste si le fichier est protégé avant de l'effacer...

Et maintenant quelques exemples: Soit une disquette double face, 80 pistes, 16 secteurs:

DUMP1 (après écriture d'un fichier):

```
Drive A V3 (Mst)                XX/XX/XX
FICHIER01.COM  16
2437 sectors free (D/80/16)  1 Files

Piste 20 Secteur 2 (après écriture)
00+FF008509 01005010 01D00000 00000000
10+00000000 00000000 00000000 0000F8FF
20+FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
30+FFFFFFFF FFFFFFFF B06DFFFF FFFFetc.

Piste 20 Secteur 3 (après écriture)
00+FF00000A 00005010 01D00000 00000000
10+FFFFFFFF FFFFFFFF FFFFFFFF FFFFetc.

Piste 20 Secteur 4 (après écriture)
00+00002000 00000000 00000000 00000000
10+46494348 49455230 31434F4D 06041040
     F I C H  I E R 0  1 C O M
20+00000000 00000000 00000000 0000etc.
```

DUMP2 (après suppression du fichier):

```
Drive A V3 (Mst)                XX/XX/XX

2453 sectors free (D/80/16)  0 Files

Piste 20 Secteur 2 (après suppression)
00+FF009509 00005010 01D00000 00000000
10+00000000 00000000 00000000 F8FFFFFF
20+FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
30+FFFFFFFF FFFFFFFF B06DFFFF FFFFetc.

Piste 20 Secteur 3 (après suppression)
00+FF00000A 00005010 01D00000 00000000
10+FFFFFFFF FFFFFFFF FFFFFFFF FFFFetc.

Piste 20 Secteur 4 (après suppression)
00+00001000 00000000 00000000 00000000
10+00000000 00000000 00000000 00000000

20+00000000 00000000 00000000 0000etc.
```

En gras dans les DUMPs ci-dessus, sont indiquées les différences entre la disquette après écriture du fichier et la disquette après suppression du fichier. Il s'agit, dans l'ordre, du nombre de secteurs libres, du nombre de fichiers, de la bitmap proprement dite, de l'offset de la première "entrée" libre et de la ligne d'entrée dans le catalogue, ligne sur laquelle figure le nom du fichier et les coordonnées de son descripteur principal. Il

est à noter qu'après la suppression les secteurs de bitmap et de directory sont redevenus identiques à ce qu'ils étaient avant écriture, par contre le secteur de descripteur et les 15 secteurs de data n'ont pas été supprimés (non montré).

Un peu plus de détails...

Lorsqu'il "efface" un fichier, SEDORIC ne modifie en fait que les secteurs de bitmap et de directory. Le ou les secteurs de descripteurs ainsi que les secteurs de data restent intacts... tant qu'un nouveau fichier n'est pas écrit dans les secteurs libérés. Il serait tout à fait possible d'écrire un programme UNDEL pour récupérer un fichier effacé par erreur.

Modifications dans la bitmap: pour libérer les secteurs, SEDORIC charge le descripteur principal et lit la liste des coordonnées piste/secteur de ce fichier et s'il y a lieu, des fichiers mergés qui lui sont attachés. Dans la bitmap, il inverse le bit correspondant à chacun des secteurs de data et des descripteurs de ce fichier. Finalement, il décrémente le nombre de fichiers et met à jour le nombre de secteurs libres.

Modification dans le directory: Sauf dans le cas où le fichier supprimé était le dernier de la liste, SEDORIC procède ensuite à une restructuration du secteur de directory dans lequel il a effacé une "entrée". Cette restructuration est en général très primitive: “l'entrée" supprimée est en fait écrasée par la dernière de la liste, laquelle devenue inutile est alors surchargée de zéros. Si “l'entrée" du fichier supprimé était la dernière d'un secteur de directory (la quinzième donc) elle est simplement surchargée de zéros. Enfin SEDORIC décrémente le pointeur de la prochaine "entrée" libre dans ce secteur de directory.

Après plusieurs DEL, l'ensemble des secteurs de directory est donc parsemé de trous, qui sont toujours localisés à la fin de ces secteurs de directory. Ces trous sont réutilisés en priorité lors d'un nouveau SAVE. En effet, SEDORIC cherche une place en examinant par le début la suite des divers secteurs de directory.

Récupération de secteurs de directory. La restructuration est parfois plus complexe. Avant de rendre la main, lorsque SEDORIC a effacé tout ce qu'on lui avait demandé, il calcule combien de secteurs de directory sont nécessaires, compte tenu du nombre de fichiers restants. S'il y a au moins un secteur de catalogue de plus que nécessaire, SEDORIC va chercher le dernier secteur de catalogue et en recopie les "entrées" valides dans les trous des secteurs de catalogue précédents. Il recherche ces trous en commençant par le premier secteur de directory. A l'issue de ce transfert, l'avant dernier secteur de directory devient le nouveau dernier secteur de directory, SEDORIC y écrit donc des zéros à l'endroit des coordonnées du secteur de directory suivant. Lorsque le dernier secteur de directory est entièrement vidé, il est lui-même libéré dans la bitmap (s'il s'agit d'un des secteurs 4, 7, 10, 13 ou 16 de la piste 20, seul le nombre de secteurs de directory en service est décrémenté: le secteur reste "réservé" (occupé) dans la bitmap). SEDORIC reboucle si nécessaire pour libérer un autre secteur de directory.

Encore quelques exemples:

Soit la même disquette après avoir écrit 22 fichiers dans l'ordre FICHIER01.COM à FICHIER22.COM, sachant que le quatorzième est"à cheval" sur les deux faces, que le quinzième est inscrit sur la dernière ligne d' "entrée" du premier secteur de directory et présent sur la deuxième face de la disquette et que le vingtième est "à cheval" sur les deux secteurs de bitmap. Nous allons faire subir à cette disquette un certain nombre de délétion type et examiner ce qui se passe. Pour plus de facilité la comparaison se fera toujours avec l'état initial.

DUMP3 (Etat initial: 22 fichiers):

```
Drive A V3 (Mst)              XX/XX/XX
FICHIER01.COM  16   FICHIER02.COM  16
FICHIER03.COM 100   FICHIER04.COM 100
FICHIER05.COM 100   FICHIER06.COM 100
FICHIER07.COM 100   FICHIER08.COM 100
FICHIER09.COM 100   FICHIER10.COM 100
FICHIER11.COM 100   FICHIER12.COM 100
FICHIER13.COM 100   FICHIER14.COM 100
FICHIER15.COM 100   FICHIER16.COM 100
FICHIER17.COM 100   FICHIER18.COM 100
FICHIER19.COM 100   FICHIER20.COM 100
FICHIER21.COM 100    FICHIER22.COM 100
*421 sectors free (D/80/16) 22 Files

Piste 20 Secteur 2 (Etat initial)
00+FF00A501 16005010 02D00000 00000000
10+00000000 00000000 00000000 00000000
20+00000000 00000000 00000000 0000etc.
90+00000000 00000000 00000000 00000000
A0+00000000 00000000 00000000 00000000
B0+00000000 00000000 00000000 00000000
C0+00000000 00000000 00000000 0000etc.
F0+00000000 00000000 00000000 00000000

Piste 20 Secteur 3 (Etat initial)
00+FF00A501 16005010 02D00000 00000000
10+00000000 00000000 00000000 00000000
20+00000000 00000000 000000F8 FFFFetc.

Piste 20 Secteur 4 (Etat initial)
00+14070000 00000000 00000000 00000000
10+ F I C H  I E R 0  1 C O M 06041040
20+ F I C H  I E R 0  2 C O M 07041040
30+ F I C H  I E R 0  3 C O M 08046440
40+ F I C H  I E R 0  4 C O M 0E086440
50+ F I C H  I E R 0  5 C O M 15046440
60+ F I C H  I E R 0  6 C O M 1B086440
70+ F I C H  I E R 0  7 C O M 210C6440
80+ F I C H  I E R 0  8 C O M 27106440
90+ F I C H  I E R 0  9 C O M 2E046440
A0+ F I C H  I E R 1  0 C O M 34086440
B0+ F I C H  I E R 1  1 C O M 3A0C6440
C0+ F I C H  I E R 1  2 C O M 40106440
D0+ F I C H  I E R 1  3 C O M 47046440
E0+ F I C H  I E R 1  4 C O M 4D086440
F0+ F I C H  I E R 1  5 C O M 830C6440

Piste 20 Secteur 7 (Etat initial)
00+00008000 00000000 00000000 00000000
10+ F I C H  I E R 1  6 C O M 89106440
20+ F I C H  I E R 1  7 C O M 90046440
30+ F I C H  I E R 1  8 C O M 96086440
40+ F I C H  I E R 1  9 C O M 9C0C6440
50+ F I C H  I E R 2  0 C O M A2106440
60+ F I C H  I E R 2  1 C O M A9046440
70+ F I C H  I E R 2  2 C O M AF086440
80+00000000 00000000 00000000 0000etc.
```

DUMP4 (Après DEL du premier fichier):

Drive A V3 (Mst) XX/XX/XX
FICHIER15.COM 100 FICHIER02.COM 16
FICHIER03.COM 100 FICHIER04.COM 100
FICHIER05.COM 100 FICHIER06.COM 100
FICHIER07.COM 100 FICHIER08.COM 100
FICHIER09.COM 100 FICHIER10.COM 100
FICHIER11.COM 100 FICHIER12.COM 100
FICHIER13.COM 100 FICHIER14.COM 100
FICHIER16.COM 100 FICHIER17.COM 100
FICHIER18.COM 100 FICHIER19.COM 100
FICHIER20.COM 100 FICHIER21.COM 100
FICHIER22.COM 100
***437** sectors free (D/80/16) **21** Files

Piste 20 Secteur 2 (DEL 1er fichier)
00+FF00**B5**01 **15**005010 02D00000 00000000
10+00000000 00000000 00000000 **F8FF07**00
20+00000000 00000000 00000000 0000etc.
90+00000000 00000000 00000000 00000000
A0+00000000 00000000 00000000 00000000
B0+00000000 00000000 00000000 00000000
C0+00000000 00000000 00000000 0000etc.
F0+00000000 00000000 00000000 00000000

Piste 20 Secteur 3 (DEL 1er fichier)
00+FF00A501 16005010 02D00000 00000000
10+00000000 00000000 00000000 00000000
20+00000000 00000000 000000F8 FFFFetc.

Piste 20 Secteur 4 (DEL 1er fichier)
00+1407**F0**00 00000000 00000000 00000000
10+ F I C H I E R **1** **5** C O M **830C64**40
20+ F I C H I E R 0 2 C O M 07041040
30+ F I C H I E R 0 3 C O M 08046440
40+ F I C H I E R 0 4 C O M 0E086440
50+ F I C H I E R 0 5 C O M 15046440
60+ F I C H I E R 0 6 C O M 1B086440
70+ F I C H I E R 0 7 C O M 210C6440
80+ F I C H I E R 0 8 C O M 27106440
90+ F I C H I E R 0 9 C O M 2E046440
A0+ F I C H I E R 1 0 C O M 34086440
B0+ F I C H I E R 1 1 C O M 3A0C6440
C0+ F I C H I E R 1 2 C O M 40106440
D0+ F I C H I E R 1 3 C O M 47046440
E0+ F I C H I E R 1 4 C O M 4D086440
F0+**00000000 00000000 00000000 00000000**

Piste 20 Secteur 7 (DEL 1er fichier)
00+00008000 00000000 00000000 00000000
10+ F I C H I E R 1 6 C O M 89106440
20+ F I C H I E R 1 7 C O M 90046440
30+ F I C H I E R 1 8 C O M 96086440
40+ F I C H I E R 1 9 C O M 9C0C6440
50+ F I C H I E R 2 0 C O M A2106440
60+ F I C H I E R 2 1 C O M A9046440
70+ F I C H I E R 2 2 C O M AF086440
80+00000000 00000000 00000000 0000etc.

Les octets qui diffèrent ont été indiqués en gras. Lorsque le premier fichier est supprimé, il est remplacé

dans le directory par le dernier fichier du premier secteur de directory. Les nombres de secteurs libres (#01B5 = 437) et de fichiers (#15 = 21) sont mis à jour. Dans la bitmap proprement dite, c'est à dire à partir de l'octet numéro #10 du deuxième secteur de la piste 20, chaque piste (ici 16 secteurs) occupe deux octets, en commençant par la piste numéro zéro. Le secteur descripteur du premier fichier se trouvait au quatrième secteur de la piste 6 et les secteurs de data correspondants continuaient jusqu'à troisième secteur de la piste 7. Vous pourrez vérifier que 16 secteurs ont bien été libérés (000000->F8FF07). Dans le premier secteur de directory, le pointeur de première "entrée" libre indique #F0 et sur la ligne #F0, l'ancienne "entrée" du quinzième fichier est surchargée de zéros.

Si maintenant on supprime le quatorzième fichier, qui est "à cheval" sur les deux faces, ont obtiendra des modifications analogues, mais les 100 secteurs libérés dans la bitmap s'échelonneront du huitième secteur de la piste numéro #4D (77) au secteur numéro #0B (11) de la piste numéro #83 (piste numéro 3 de la deuxième face) (DUMP5). Essayez de comprendre ce qui se passe lorsqu'on supprime le quinzième fichier, qui est répertorié sur la dernière ligne d'entrée du premier secteur de directory et qui se trouve sur la deuxième face de la disquette (DUMP6). Dans les deux cas, les octets qui différent de l'état initial de la disquette (DUMP3) sont indiqués en gras.

<u>DUMP5 (après DEL du 14e fichier):</u>

```
Drive A V3 (Mst)              XX/XX/XX
FICHIER01.COM  16   FICHIER02.COM  16
FICHIER03.COM 100   FICHIER04.COM 100
FICHIER05.COM 100   FICHIER06.COM 100
FICHIER07.COM 100   FICHIER08.COM 100
FICHIER09.COM 100   FICHIER10.COM 100
FICHIER11.COM 100   FICHIER12.COM 100
FICHIER13.COM 100   FICHIER15.COM 100
FICHIER16.COM 100   FICHIER17.COM 100
FICHIER18.COM 100   FICHIER19.COM 100
FICHIER20.COM 100   FICHIER21.COM 100
FICHIER22.COM 100
*521 sectors free (D/80/16) 21 Files

Piste 20 Secteur 2 (DEL 14e fichier)
00+FF000902 15005010 02D00000 0000etc.
A0+00000000 00000000 000080FF FFFFFFFF
B0+FFFFFFFF FFFFFF07 00000000 00000000
C0+00000000 00000000 00000000 00000etc.

Piste 20 Secteur 4 (DEL 14e fichier)
00+1407F000 00000000 00000000 00000000
10+ F I C H  I E R 0  1 C O M 06041040
.../...
C0+ F I C H  I E R 1  2 C O M 40106440
D0+ F I C H  I E R 1  3 C O M 47046440
E0+ F I C H  I E R 1  5 C O M 830C6440
F0+00000000 00000000 00000000 00000000
```

<u>DUMP6 (après DEL du 15e fichier):</u>

```
Drive A V3 (Mst)              XX/XX/XX
FICHIER01.COM  16   FICHIER02.COM  16
FICHIER03.COM 100   FICHIER04.COM 100
FICHIER05.COM 100   FICHIER06.COM 100
FICHIER07.COM 100   FICHIER08.COM 100
FICHIER09.COM 100   FICHIER10.COM 100
FICHIER11.COM 100   FICHIER12.COM 100
FICHIER13.COM 100   FICHIER14.COM 100
FICHIER16.COM 100   FICHIER17.COM 100
FICHIER18.COM 100   FICHIER19.COM 100
FICHIER20.COM 100   FICHIER21.COM 100
FICHIER22.COM 100
*521 sectors free (D/80/16) 21 Files

Piste 20 Secteur 2 (DEL 15e fichier)
00+FF000902 15005010 02D00000 0000etc.
A0+00000000 00000000 00000000 00000000
B0+00000000 000000F8 FFFFFFFF FFFFFFFF
C0+FFFFFF7F 00000000 00000000 0000etc.

Piste 20 Secteur 4 (DEL 15e fichier)
00+1407F000 00000000 00000000 00000000
10+ F I C H  I E R 0  1 C O M 06041040
.../...
C0+ F I C H  I E R 1  2 C O M 40106440
D0+ F I C H  I E R 1  3 C O M 47046440
E0+ F I C H  I E R 1  4 C O M 4D086440
F0+00000000 00000000 00000000 00000000
```

Maintenant on supprime le vingtième fichier, qui est répertorié dans le deuxième secteur de directory, qui est lui aussi présent sur la deuxième face de la disquette, mais qui est "à cheval" sur les 2 secteurs de bitmap (DUMP7). Les différences avec l'état initial de la disquette sont toujours indiquées en gras

DUMP7 (après DEL du 20e fichier):

```
Drive A V3 (Mst)                 XX/XX/XX
FICHIER01.COM 16   FICHIER02.COM  16
FICHIER03.COM 100  FICHIER04.COM 100
FICHIER05.COM 100  FICHIER06.COM 100
FICHIER07.COM 100   FICHIER08.COM 100
FICHIER09.COM 100   FICHIER10.COM 100
FICHIER11.COM 100  FICHIER12.COM 100
FICHIER13.COM 100   FICHIER14.COM 100
FICHIER15.COM 100   FICHIER16.COM 100
FICHIER17.COM 100   FICHIER18.COM 100
FICHIER19.COM 100   FICHIER22.COM 100
FICHIER21.COM 100
*521 sectors free (D/80/16) 21 Files
```

```
Piste 20 Secteur 2 (DEL 20e fichier)
00+FF000902 15005010 02D00000 00000000
10+00000000 00000000 00000000 00000000
20+00000000 00000000 00000000 0000etc.
90+00000000 00000000 00000000 00000000
A0+00000000 00000000 00000000 00000000
B0+00000000 00000000 00000000 00000000
C0+00000000 00000000 00000000 0000etc.
F0+00000000 0080FFFF FFFFFFFF FFFFFFFF
```

```
Piste 20 Secteur 3 (DEL 20e fichier)
00+FF000902 15005010 02D00000 00000000
10+FFFF0700 00000000 00000000 00000000
20+00000000 00000000 000000F8 FFFFetc.
```

```
Piste 20 Secteur 4 (DEL 20e fichier)
00+14070000 00000000 00000000 00000000
10+ F I C H  I E R 0  1 C O M 06041040
20+ F I C H  I E R 0  2 C O M 07041040
.../...
F0+ F I C H  I E R 1  5 C O M 830C6440
```

```
Piste 20 Secteur 7 (DEL 20e fichier)
00+00007000 00000000 00000000 00000000
10+ F I C H  I E R 1  6 C O M 89106440
20+ F I C H  I E R 1  7 C O M 90046440
30+ F I C H  I E R 1  8 C O M 96086440
40+ F I C H  I E R 1  9 C O M 9C0C6440
50+ F I C H  I E R 2  2 C O M AF086440
60+ F I C H  I E R 2  1 C O M A9046440
70+00000000 00000000 00000000 00000000
80+00000000 00000000 00000000 0000etc.
```

Pour terminer, nous allons voir ce qui se passe lorsqu'on effectue des suppressions multiples (le douzième fichier, puis le deuxième fichier qui sont répertoriés dans le premier secteur de directory et qui se trouvent tous les deux sur la première face de la disquette), ou des suppression suivies de l'écriture de nouveaux fichiers. Dans ce dernier exemple, on supprimera le douzième (premier secteur de directory, première face de la disquette, premier secteur de bitmap), puis le vingtième fichier (deuxième secteur de directory, deuxième face de la disquette et "à cheval" sur les 2 secteurs de bitmap), puis on écrira un vingt-troisième et un vingt-quatrième fichiers (DUMP9).

DUMP8 (après DEL multiples):

```
Drive A V3 (Mst)               XX/XX/XX
FICHIER01.COM  16   FICHIER14.COM 100
FICHIER03.COM 100   FICHIER04.COM 100
FICHIER05.COM 100   FICHIER06.COM 100
FICHIER07.COM 100   FICHIER08.COM 100
FICHIER09.COM 100   FICHIER10.COM 100
FICHIER11.COM 100   FICHIER15.COM 100
FICHIER13.COM 100   FICHIER16.COM 100
FICHIER17.COM 100   FICHIER18.COM 100
FICHIER19.COM 100   FICHIER20.COM 100
FICHIER21.COM 100   FICHIER22.COM 100
*537 sectors free (D/80/16) 20 Files
```

```
Piste 20 Secteur 2 (DEL multiple)
00+FF001902 14005010 02D00000 00000000
10+00000000 00000000 00000000 0000F8FF
20+07000000 00000000 00000000 0000etc.
90+0080FFFF FFFFFFFF FFFFFFFF FFFF0700
A0+00000000 00000000 00000000 0000etc.
```

DUMP9 (après DEL puis SAVE):

```
Drive A V3 (Mst)                   XX/XX/XX
FICHIER01.COM  16   FICHIER02.COM  16
FICHIER03.COM 100   FICHIER04.COM 100
FICHIER05.COM 100   FICHIER06.COM 100
FICHIER07.COM 100   FICHIER08.COM 100
FICHIER09.COM 100   FICHIER10.COM 100
FICHIER11.COM 100   FICHIER15.COM 100
FICHIER13.COM 100   FICHIER14.COM 100
FICHIER23.COM 100   FICHIER16.COM 100
FICHIER17.COM 100   FICHIER18.COM 100
FICHIER19.COM 100   FICHIER22.COM 100
FICHIER21.COM 100   FICHIER24.COM 100
*421 sectors free (D/80/16) 22 Files
```

```
Piste 20 Secteur 2 (DEL puis SAVE)
00+FF00A501 16005010 02D00000 00000000
10+00000000 00000000 00000000 00000000
20+00000000 00000000 00000000 0000etc.
90+00000000 00000000 00000000 00000000
A0+00000000 00000000 00000000 0000etc.
```

```
Piste 20 Secteur 3 (DEL multiple)
00+FF00A501 16005010 02D00000 00000000
10+00000000 00000000 000000F8 FFFFetc.

Piste 20 Secteur 4 (DEL multiple)
00+1407E000 00000000 00000000 00000000
10+ F I C H  I E R 0  1 C O M 06041040
20+ F I C H  I E R 1  4 C O M 4D086440
30+ F I C H  I E R 0  3 C O M 08046440
40+ F I C H  I E R 0  4 C O M 0E086440
50+ F I C H  I E R 0  5 C O M 15046440
60+ F I C H  I E R 0  6 C O M 1B086440
70+ F I C H  I E R 0  7 C O M 210C6440
80+ F I C H  I E R 0  8 C O M 27106440
90+ F I C H  I E R 0  9 C O M 2E046440
A0+ F I C H  I E R 1  0 C O M 34086440
B0+ F I C H  I E R 1  1 C O M 3A0C6440
C0+ F I C H  I E R 1  5 C O M 830C6440
D0+ F I C H  I E R 1  3 C O M 47046440
E0+00000000 00000000 00000000 00000000
F0+00000000 00000000 00000000 00000000

Piste 20 Secteur 7 (DEL multiple)
00+00008000 00000000 00000000 00000000
10+ F I C H  I E R 1  6 C O M 89106440
20+ F I C H  I E R 1  7 C O M 90046440
30+ F I C H  I E R 1  8 C O M 96086440
40+ F I C H  I E R 1  9 C O M 9C0C6440
50+ F I C H  I E R 2  0 C O M A2106440
60+ F I C H  I E R 2  1 C O M A9046440
70+ F I C H  I E R 2  2 C O M AF086440
80+00000000 00000000 00000000 0000etc.
```

```
Piste 20 Secteur 3 (DEL puis SAVE)
00+FF00A501 16005010 02D00000 00000000
20+00000000 00000000 000000F8 FFFFetc.

Piste 20 Secteur 4 (DEL puis SAVE)
00+14070000 00000000 00000000 00000000
10+ F I C H  I E R 0  1 C O M 06041040
20+ F I C H  I E R 0  2 C O M 07041040
30+ F I C H  I E R 0  3 C O M 08046440
40+ F I C H  I E R 0  4 C O M 0E086440
50+ F I C H  I E R 0  5 C O M 15046440
60+ F I C H  I E R 0  6 C O M 1B086440
70+ F I C H  I E R 0  7 C O M 210C6440
80+ F I C H  I E R 0  8 C O M 27106440
90+ F I C H  I E R 0  9 C O M 2E046440
A0+ F I C H  I E R 1  0 C O M 34086440
B0+ F I C H  I E R 1  1 C O M 3A0C6440
C0+ F I C H  I E R 1  5 C O M 830C6440
D0+ F I C H  I E R 1  3 C O M 47046440
E0+ F I C H  I E R 1  4 C O M 4D086440
F0+ F I C H  I E R 2  3 C O M 40106440

Piste 20 Secteur 7 (DEL puis SAVE)
00+00008000 00000000 00000000 00000000
10+ F I C H  I E R 1  6 C O M 89106440
20+ F I C H  I E R 1  7 C O M 90046440
30+ F I C H  I E R 1  8 C O M 96086440
40+ F I C H  I E R 1  9 C O M 9C0C6440
50+ F I C H  I E R 2  2 C O M AF086440
60+ F I C H  I E R 2  1 C O M A9046440
70+ F I C H  I E R 2  4 C O M A2106440
80+00000000 00000000 00000000 0000etc.
```

Commentaires sur le DUMP8 (suppression multiple): Le DIR et l'examen du premier secteur de directory montrent que le douzième fichier a été supprimé et que son "entrée" dans le premier secteur de directory a été remplacée par celle du quinzième fichier. La quinzième et dernière "entrée", devenue inutile a été surchargée avec des zéros. Puis le deuxième fichier a été supprimé et son "entrée" a été remplacée par la dernière "entrée" valide, c'est à dire celle du quatorzième fichier, laquelle devenue inutile a été surchargée avec des zéros. On peut suivre les divers ajustements que cela a entraîné dans le premier secteur de bitmap. Le nombre de secteurs libres a été ajusté à #219 (537). Le nombre de fichiers à été réduit à #14 (20). Des secteurs ont été libérés dans les pistes numéro 7, 8, 64 à 71. Le deuxième secteur de bitmap n'a pas été affecté. Le deuxième secteur de directory non plus.

Commentaires sur le DUMP9 (suppression puis écriture): Le douzième fichier a été supprimé et son "entrée" dans le premier secteur de directory a été remplacée par celle du quinzième fichier. La quinzième et dernière "entrée", devenue inutile a été surchargée avec des zéros. Puis le vingtième fichier a été supprimé et son "entrée", dans le deuxième secteur de directory, a été remplacée par la dernière "entrée" valide, c'est à dire celle du vingt-deuxième fichier, laquelle devenue inutile a été surchargée avec des zéros. Puis le vingt-troisième fichier a été écrit et son entrée a été mise à la première place libre, c'est à dire à l'ancienne place du quinzième fichier. Enfin le vingt-quatrième fichier a été écrit et son entrée a été mise à la place libre suivante, l'ancienne place du vingt-deuxième fichier. On l'absence de changements dans le premier secteur de bitmap, en effet, dans ce cas précis, le nombre de fichiers est le même, ainsi que leur taille. Par suite, aucun secteur n'a été libéré ou occupé. En fait, 200 secteurs ont été libérés et les mêmes ont été re-occupés, mais le bilan est inchangé. Seuls les premier et deuxième secteurs de directory ont été modifiés!

ANNEXE n° 10
L’ EPROM du MICRODISC

Ce document est de Fabrice Francès et a été chargé à l'URL:
http://www.ensica.fr/oric/HARDWARE/Eprom.lst

```
E000  4C C2 E5  |  JMP $E5C2     ; initializes some parameters
E003  4C 0C E2  |  JMP $E20C     ; FDC routine
E006  4C 34 EB  |  JMP $EB34
E009  4C CE E4  |  JMP $E4CE     ; loads a file
E00C  4C D3 EA  |  JMP $EAD3     ; searches a file
E00F  4C DE E4  |  JMP $E4DE     ; error "File not found"
E012  4C FC EA  |  JMP $EAFC
E015  4C 17 E1  |  JMP $E117     ; lets the user type a command in TIB
E018  4C 1F E1  |  JMP $E11F     ; waits for a keypress
E01B  4C DC E7  |  JMP $E7DC     ; error routine
E01E  4C 16 E8  |  JMP $E816     ; dummy, points to a RTS
E021  4C 17 E8  |  JMP $E817     ; writes a sector
E024  4C 25 E8  |  JMP $E825     ; reads sector
E027  4C 2B E8  |  JMP $E82B     ; reads boot sector
E02A  4C 46 E8  |  JMP $E846     ; checks drive number
E02D  4C 54 E8  |  JMP $E854     ; prints string pointed by ($0C)
E030  4C 80 E9  |  JMP $E980     ; points to next directory entry
E033  4C 72 E8  |  JMP $E872     ; reads system parameters from boot sector
E036  4C 81 E8  |  JMP $E881     ; writes system parameters to boot sector
E039  4C 93 E8  |  JMP $E893     ; adds a directory entry (not used)
E03C  4C B7 E8  |  JMP $E8B7
E03F  4C C5 E8  |  JMP $E8C5
E042  4C F6 E8  |  JMP $E8F6
E045  4C 50 E9  |  JMP $E950
E048  4C A8 E9  |  JMP $E9A8     ; reads boot sector
E04B  4C CA E9  |  JMP $E9CA
E04E  4C B8 EA  |  JMP $EAB8     ; limits a char to alphanumeric
E051  4C EA EA  |  JMP $EAEA
E054  4C 27 E1  |  JMP $E127     ; new line
E057  4C 2E E1  |  JMP $E12E     ; prints char
E05A  4C 70 E0  |  JMP $E070     ; calls a routine in ROM Basic
E05D  4C 6A E1  |  JMP $E16A     ; interprets a decimal or hex number
E060  4C 63 E1  |  JMP $E163     ; reads a non-blank char
E063  4C C7 E4  |  JMP $E4C7
E066  4C 00 00  |  JMP $0000
E069  4C 00 00  |  JMP $0000
E06C  4C 45 EB  |  JMP $EB45     ; checks no '?' wildcard is used

E06F: 00

;******************************************************************************
; calls a routine in Basic ROM:
; grabs the two addresses after the JSR and selects one depending on the version
;******************************************************************************

E070  08        |  PHP
E071  48        |  PHA
```

```
E072  8A        |  TXA
E073  48        |  PHA
E074  98        |  TYA
E075  48        |  PHA
E076  BA        |  TSX
E077  BD 05 01  |  LDA $0105,X
E07A  18        |  CLC
E07B  85 0E     |  STA $0E
E07D  69 04     |  ADC #$04
E07F  9D 05 01  |  STA $0105,X
E082  BD 06 01  |  LDA $0106,X
E085  85 0F     |  STA $0F
E087  69 00     |  ADC #$00
E089  9D 06 01  |  STA $0106,X
E08C  A0 01     |  LDY #$01
E08E  AD 07 C0  |  LDA $C007
E091  F0 02     |  BEQ $E095
E093  A0 03     |  LDY #$03
E095  B1 0E     |  LDA ($0E),Y
E097  8D 85 04  |  STA $0485
E09A  C8        |  INY
E09B  B1 0E     |  LDA ($0E),Y
E09D  8D 86 04  |  STA $0486
E0A0  A9 06     |  LDA #$06
E0A2  8D 81 04  |  STA $0481
E0A5  68        |  PLA
E0A6  A8        |  TAY
E0A7  68        |  PLA
E0A8  AA        |  TAX
E0A9  68        |  PLA
E0AA  28        |  PLP
E0AB  4C 90 04  |  JMP $0490


;*****************************************************************************
;       NMI : handler
;*****************************************************************************

E0AE  48        |  PHA
E0AF  AD 81 04  |  LDA $0481
E0B2  48        |  PHA
E0B3  AD 85 04  |  LDA $0485
E0B6  48        |  PHA
E0B7  AD 86 04  |  LDA $0486
E0BA  48        |  PHA
E0BB  AD 80 04  |  LDA $0480
E0BE  29 FE     |  AND #$FE
E0C0  8D 80 04  |  STA $0480
E0C3  8D 14 03  |  STA $0314
E0C6  A9 00     |  LDA #$00
E0C8  8D 85 04  |  STA $0485
E0CB  A9 00     |  LDA #$00
E0CD  8D 86 04  |  STA $0486
E0D0  A9 06     |  LDA #$06
E0D2  8D 81 04  |  STA $0481
E0D5  20 90 04  |  JSR $0490
E0D8  68        |  PLA
E0D9  8D 86 04  |  STA $0486
```

```
E0DC  68        |  PLA
E0DD  8D 85 04  |  STA $0485
E0E0  68        |  PLA
E0E1  8D 81 04  |  STA $0481
E0E4  68        |  PLA
E0E5  40        |  RTI

;*****************************************************************************
;           IRQ wrapper
;        -> switch to ROM and exec the Basic IRQ handler
;*****************************************************************************

E0E6  48        |  PHA
E0E7  8A        |  TXA
E0E8  48        |  PHA
E0E9  AD 81 04  |  LDA $0481
E0EC  48        |  PHA
E0ED  AD 85 04  |  LDA $0485
E0F0  48        |  PHA
E0F1  AD 86 04  |  LDA $0486
E0F4  48        |  PHA
E0F5  A9 8A     |  LDA #$8A
E0F7  8D 85 04  |  STA $0485
E0FA  A9 04     |  LDA #$04
E0FC  8D 86 04  |  STA $0486
E0FF  A9 06     |  LDA #$06
E101  8D 81 04  |  STA $0481
E104  20 90 04  |  JSR $0490
E107  68        |  PLA
E108  8D 86 04  |  STA $0486
E10B  68        |  PLA
E10C  8D 85 04  |  STA $0485
E10F  68        |  PLA
E110  8D 81 04  |  STA $0481
E113  68        |  PLA
E114  AA        |  TAX
E115  68        |  PLA
E116  40        |  RTI

;*****************************************************************************
; waits for the user to type a command in TIB
;
E117  20 5A E0  |  JSR $E05A
E11A  A2 C5 92 C5
E11E  60        |  RTS

;*****************************************************************************
; waits for a keypress, ascii code returned in A
;
E11F  20 5A E0  |  JSR $E05A
E122  F8 C5 E8 C5
E126  60        |  RTS

;****************************************************************************
; prints carriage return + line feed
;
E127  A9 0D     |  LDA #$0D
E129  20 2E E1  |  JSR $E12E
```

```
E12C  A9 0A     |  LDA #$0A
;
; prints char
;
E12E  08        |  PHP
E12F  8E 51 C1  |  STX $C151
E132  AA        |  TAX
E133  48        |  PHA
E134  A5 0C     |  LDA $0C
E136  48        |  PHA
E137  A5 0D     |  LDA $0D
E139  48        |  PHA
E13A  20 5A E0  |  JSR $E05A     ; calls Basic's output routine
E13D  3F F7 7C F7
E141  68        |  PLA
E142  85 0D     |  STA $0D
E144  68        |  PLA
E145  85 0C     |  STA $0C
E147  68        |  PLA
E148  AE 51 C1  |  LDX $C151
E14B  28        |  PLP
E14C  60        |  RTS

;*****************************************************************************
; prints a hex byte
;
E14D  48        |  PHA
E14E  4A        |  LSR
E14F  4A        |  LSR
E150  4A        |  LSR
E151  4A        |  LSR
E152  20 56 E1  |  JSR $E156
E155  68        |  PLA
E156  29 0F     |  AND #$0F
E158  09 30     |  ORA #$30
E15A  C9 3A     |  CMP #$3A
E15C  90 D0     |  BCC $E12E
E15E  69 06     |  ADC #$06
E160  D0 CC     |  BNE $E12E
;*****************************************************************************
; reads next non-blank char

E162  C8        |  INY

E163  B1 E9     |  LDA ($E9),Y
E165  C9 20     |  CMP #$20
E167  F0 F9     |  BEQ $E162
E169  60        |  RTS

;*****************************************************************************
; interprets decimal and hexadecimal numbers

E16A  A9 00     |  LDA #$00      ; initializes the number read
E16C  8D 45 C1  |  STA $C145
E16F  8D 46 C1  |  STA $C146
E172  B1 E9     |  LDA ($E9),Y   ; skips any blanks
E174  C8        |  INY
E175  C9 20     |  CMP #$20
```

```
E177  F0 F9     |  BEQ $E172
E179  C9 23     |  CMP #$23
E17B  D0 24     |  BNE $E1A1     ; is it a '#' ?

E17D  B1 E9     |  LDA ($E9),Y   ; yes, reads the hex number
E17F  20 F1 E1  |  JSR $E1F1
E182  90 1B     |  BCC $E19F     ; is it a hex digit ?
E184  C8        |  INY                ; yes, computes the hex number read so far
E185  A2 04     |  LDX #$04
E187  0E 45 C1  |  ASL $C145
E18A  2E 46 C1  |  ROL $C146
E18D  CA        |  DEX
E18E  D0 F7     |  BNE $E187
E190  18        |  CLC
E191  6D 45 C1  |  ADC $C145
E194  8D 45 C1  |  STA $C145
E197  90 E4     |  BCC $E17D
E199  EE 46 C1  |  INC $C146
E19C  4C 7D E1  |  JMP $E17D
E19F  38        |  SEC           ; no, returns C=1
E1A0  60        |  RTS

E1A1  88        |  DEY           ; first char was not a '!', goes back on it
E1A2  20 E6 E1  |  JSR $E1E6     ; and reads a decimal number
E1A5  90 F9     |  BCC $E1A0
E1A7  C8        |  INY
E1A8  48        |  PHA
E1A9  AD 46 C1  |  LDA $C146
E1AC  48        |  PHA
E1AD  AD 45 C1  |  LDA $C145
E1B0  0E 45 C1  |  ASL $C145
E1B3  2E 46 C1  |  ROL $C146
E1B6  0E 45 C1  |  ASL $C145
E1B9  2E 46 C1  |  ROL $C146
E1BC  18        |  CLC
E1BD  6D 45 C1  |  ADC $C145
E1C0  8D 45 C1  |  STA $C145
E1C3  68        |  PLA
E1C4  6D 46 C1  |  ADC $C146
E1C7  8D 46 C1  |  STA $C146
E1CA  0E 45 C1  |  ASL $C145
E1CD  2E 46 C1  |  ROL $C146
E1D0  68        |  PLA
E1D1  18        |  CLC
E1D2  6D 45 C1  |  ADC $C145
E1D5  8D 45 C1  |  STA $C145
E1D8  90 03     |  BCC $E1DD
E1DA  EE 46 C1  |  INC $C146
E1DD  B1 E9     |  LDA ($E9),Y
E1DF  20 E6 E1  |  JSR $E1E6
E1E2  B0 C3     |  BCS $E1A7
E1E4  38        |  SEC
E1E5  60        |  RTS

; checks for a decimal digit: returns C=1 if success
;
E1E6  38        |  SEC
E1E7  E9 30     |  SBC #$30
```

```
E1E9  90 04     |  BCC $E1EF
E1EB  C9 0A     |  CMP #$0A
E1ED  90 F5     |  BCC $E1E4
E1EF  18        |  CLC
E1F0  60        |  RTS

; checks for a hex digit: returns C=1 if success
;
E1F1  20 E6 E1  |  JSR $E1E6
E1F4  B0 EE     |  BCS $E1E4
E1F6  E9 06     |  SBC #$06
E1F8  C9 10     |  CMP #$10
E1FA  B0 F3     |  BCS $E1EF
E1FC  C9 09     |  CMP #$09
E1FE  60        |  RTS

;*******************************************************************************
; switch to Basic (no return)
;
E1FF  20 5A E0  |  JSR $E05A
E202  A3 C4 96 C4

;*******************************************************************************
;  write sector command
;*******************************************************************************
E206  A2 A0     |  LDX #$A0
E208  D0 02     |  BNE $E20C
;*******************************************************************************
;  read sector command
;*******************************************************************************
E20A  A2 80     |  LDX #$80
;*******************************************************************************
;  FDC routine: command specified in register X
;*******************************************************************************
E20C  20 E3 E3  |  JSR $E3E3     ; disables timer1 interrupts
E20F  20 1C E2  |  JSR $E21C     ; the FDC routine itself
E212  08        |  PHP
E213  8A        |  TXA
E214  48        |  PHA
E215  20 EB E3  |  JSR $E3EB     ; enables timer1 interrupts
E218  68        |  PLA
E219  AA        |  TAX
E21A  28        |  PLP
E21B  60        |  RTS

;*******************************************************************************
;  FDC routine heart: command specified in register X
;  the routine may call itself recursively,
;  thus callers have to save and restore some global variables (C005, C008,...)
;*******************************************************************************
E21C  8E 05 C0  |  STX $C005
E21F  48        |  PHA
E220  98        |  TYA
E221  48        |  PHA
E222  A9 00     |  LDA #$00
E224  8D FE 04  |  STA $04FE
E227  A9 07     |  LDA #$07
E229  8D 08 C0  |  STA $C008
```

```
E22C  20 A2 E2  |  JSR $E2A2     ; recognizes and executes the command
E22F  F0 16     |  BEQ $E247     ; error ?
E231  A8        |  TAY           ; yes...
E232  6A        |  ROR
E233  B0 55     |  BCS $E28A     ; busy ?
E235  A9 20     |  LDA #$20
E237  2C 05 C0  |  BIT $C005
E23A  10 15     |  BPL $E251     ; was it a type I command ?
E23C  50 29     |  BVC $E267     ; or a type II command ?
E23E  D0 4A     |  BNE $E28A     ; or else a read/write track command ?
E240  A9 10     |  LDA #$10
E242  2C 05 C0  |  BIT $C005
E245  F0 3B     |  BEQ $E282     ; or else a Read address id command ?
                                 ; no, just a Force Interrupt...
E247  A2 00     |  LDX #$00      ; forgets the error
E249  18        |  CLC
E24A  8E FE 04  |  STX $04FE
E24D  68        |  PLA
E24E  A8        |  TAY
E24F  68        |  PLA
E250  60        |  RTS

;*******************************************************************************
; got an error in a type I command...
;
E251  98        |  TYA
E252  29 18     |  AND #$18
E254  F0 F1     |  BEQ $E247     ; takes care of seek and crc errors only
E256  C0 18     |  CPY #$18
E258  F0 30     |  BEQ $E28A     ; returns error #1 if both seek and crc errors
E25A  AD 05 C0  |  LDA $C005     ; so, only one of these...
E25D  C9 20     |  CMP #$20
E25F  B0 29     |  BCS $E28A     ; returns error #1 if step command
E261  C9 10     |  CMP #$10
E263  90 1D     |  BCC $E282     ; but retries if Restore track 0
E265  B0 10     |  BCS $E277

;*******************************************************************************
; got an error in a type II command...
;
E267  98        |  TYA
E268  29 40     |  AND #$40
E26A  D0 1E     |  BNE $E28A     ; returns error #1 if Write protect flag
E26C  C0 10     |  CPY #$10
E26E  90 12     |  BCC $E282     ; retries if CRC error (or lost data)
E270  AD 05 C0  |  LDA $C005
E273  29 10     |  AND #$10
E275  D0 D0     |  BNE $E247     ; forgets a record not found in multiple sectors
                                 ; operations
                                 ; so, a record was not found when reading
E277  AC 05 C0  |  LDY $C005
E27A  20 64 E3  |  JSR $E364     ; read first address id encountered
E27D  8C 05 C0  |  STY $C005
E280  B0 05     |  BCS $E287     ; can't even read an address id ? gives up...
E282  CE 08 C0  |  DEC $C008     ; decrements retry counter and tries again
E285  10 A5     |  BPL $E22C
E287  20 B2 E3  |  JSR $E3B2     ; restores track 0
```

```
E28A  A2 01     |  LDX #$01      ; returns an error 1
E28C  38        |  SEC
E28D  B0 BE     |  BCS $E24D


;*****************************************************************************
;  type I commands
;
E28F  C0 20     |  CPY #$20
E291  B0 29     |  BCS $E2BC     ; step commands ? issue them ...
E293  C0 10     |  CPY #$10
E295  90 25     |  BCC $E2BC     ; restore track 0 command ? issue it...
E297  AD 01 C0  |  LDA $C001     ; no, then it is a seek command
E29A  29 7F     |  AND #$7F
E29C  8D 13 03  |  STA $0313     ; programs the track wanted
E29F  4C BC E2  |  JMP $E2BC

;*****************************************************************************
; updates the track register if needed, then recognizes the command
;
E2A2  AC 05 C0  |  LDY $C005
E2A5  20 37 E3  |  JSR $E337     ; updates the track register if needed
E2A8  B0 7B     |  BCS $E325
                                 ; now, recognizes the command:
E2AA  A9 20     |  LDA #$20
E2AC  2C 05 C0  |  BIT $C005
E2AF  10 DE     |  BPL $E28F     ; type I commands ?
E2B1  50 10     |  BVC $E2C3     ; type II commands ?
E2B3  D0 67     |  BNE $E31C     ; read/write track commands ?
E2B5  A9 10     |  LDA #$10
E2B7  2C 05 C0  |  BIT $C005
E2BA  F0 2A     |  BEQ $E2E6     ; read address id command ?
                                 ; no, so it is a force interrupt command

;*****************************************************************************
; issues the FDC command and waits for its completion (interrupt raised)
; the interrupt handler will return to the caller routine
;
E2BC  20 93 E3  |  JSR $E393     ; issues the command
E2BF  18        |  CLC
E2C0  58        |  CLI
E2C1  90 FE     |  BCC $E2C1     ; waits

;*****************************************************************************
;  type II commands : read or write a sector
;
E2C3  AD 01 C0  |  LDA $C001
E2C6  29 7F     |  AND #$7F
E2C8  EA        |  NOP
E2C9  EA        |  NOP
E2CA  CD 11 03  |  CMP $0311
E2CD  F0 11     |  BEQ $E2E0     ; is the head already on the right track ?
                                 ; no, seeks the right track first
E2CF  AD 08 C0  |  LDA $C008
E2D2  A2 1C     |  LDX #$1C
E2D4  20 1C E2  |  JSR $E21C
E2D7  8D 08 C0  |  STA $C008
E2DA  8C 05 C0  |  STY $C005
```

```
E2DD  B0 4E     |  BCS $E32D
E2DF  EA        |  NOP
                                    ; ok, the head is on the right track
E2E0  AD 02 C0  |  LDA $C002
E2E3  8D 12 03  |  STA $0312        ; programs the wanted sector
E2E6  98        |  TYA
E2E7  29 20     |  AND #$20
E2E9  D0 1A     |  BNE $E305        ; write sector command ?
                                    ; no
E2EB  20 93 E3  |  JSR $E393        ; issues the read sector command
E2EE  58        |  CLI              ; and gets the bytes,
E2EF  AD 18 03  |  LDA $0318        ; the final interrupt will exit from here
E2F2  30 FB     |  BMI $E2EF
E2F4  AD 13 03  |  LDA $0313
E2F7  91 FE     |  STA ($FE),Y
E2F9  C8        |  INY
E2FA  D0 F3     |  BNE $E2EF
E2FC  E6 FF     |  INC $FF
E2FE  D0 EF     |  BNE $E2EF
E300  F0 18     |  BEQ $E31A
E302  EA        |  NOP
E303  EA        |  NOP
E304  EA        |  NOP

E305  20 93 E3  |  JSR $E393        ; issues the write sector command
E308  58        |  CLI              ; and sends the bytes,
E309  AD 18 03  |  LDA $0318        ; the final interrupt will exit from here
E30C  30 FB     |  BMI $E309
E30E  B1 FE     |  LDA ($FE),Y
E310  8D 13 03  |  STA $0313
E313  C8        |  INY
E314  D0 F3     |  BNE $E309
E316  E6 FF     |  INC $FF
E318  D0 EF     |  BNE $E309
E31A  F0 FE     |  BEQ $E31A


;*******************************************************************************
; read/write track commands
; handles them like read/write sector commands
;
E31C  AD 05 C0  |  LDA $C005
E31F  29 10     |  AND #$10
E321  F0 C8     |  BEQ $E2EB
E323  D0 E0     |  BNE $E305

;*******************************************************************************
; address id read failed, what now ?
; the recursivity bug shows here :
; the JSR never returns if the restore track 0 fails and the stack fills up !
;
E325  20 B2 E3  |  JSR $E3B2        ; restores track 0
E328  AD FE 04  |  LDA $04FE        ; and returns status of the previous command...
E32B  58        |  CLI
E32C  60        |  RTS

;*******************************************************************************
; seek track command failed, returns interesting bits of the status
```

```
;
E32D  AD FE 04  |  LDA $04FE
E330  29 BB     |  AND #$BB
E332  8D FE 04  |  STA $04FE
E335  58        |  CLI
E336  60        |  RTS

;*****************************************************************************
; updates the track register if needed (i.e the selected drive/side changes)
;
E337  AD 00 C0  |  LDA $C000
E33A  29 03     |  AND #$03
E33C  AA        |  TAX
E33D  BD F3 E3  |  LDA $E3F3,X
E340  2C 01 C0  |  BIT $C001
E343  10 02     |  BPL $E347
E345  09 10     |  ORA #$10
E347  8D 14 03  |  STA $0314     ; programs drive and side numbers
E34A  AA        |  TAX
E34B  AD 80 04  |  LDA $0480
E34E  8E 80 04  |  STX $0480
E351  29 7E     |  AND #$7E
E353  85 FE     |  STA $FE
E355  8A        |  TXA
E356  29 7E     |  AND #$7E
E358  C5 FE     |  CMP $FE
E35A  F0 31     |  BEQ $E38D     ; were the drive/side numbers the same ?
                                 ; no, checks the drive
E35C  C0 10     |  CPY #$10      ; unless it is a seek command
                                 ; (no need to move twice)
E35E  90 2D     |  BCC $E38D
E360  C0 F0     |  CPY #$F0      ; or a format command
E362  F0 29     |  BEQ $E38D

E364  AD 04 C0  |  LDA $C004     ; reads the first address id encountered
E367  48        |  PHA
E368  A9 C3     |  LDA #$C3
E36A  8D 04 C0  |  STA $C004
E36D  AD 08 C0  |  LDA $C008
E370  A2 C0     |  LDX #$C0
E372  20 1C E2  |  JSR $E21C
E375  8D 08 C0  |  STA $C008
E378  68        |  PLA
E379  8D 04 C0  |  STA $C004
E37C  8C 05 C0  |  STY $C005
E37F  AD FE 04  |  LDA $04FE
E382  D0 0B     |  BNE $E38F
E384  AD 12 03  |  LDA $0312     ; gets the track number
E387  EA        |  NOP
E388  EA        |  NOP
E389  EA        |  NOP
E38A  8D 11 03  |  STA $0311     ; and updates the track register
E38D  18        |  CLC
E38E  60        |  RTS
E38F  38        |  SEC
E390  60        |  RTS
E391  EA        |  NOP
E392  EA        |  NOP
```

```
;******************************************************************************
; issue the effective FDC command specified in Y
;
E393  78        |  SEI
E394  8C 05 C0  |  STY $C005
E397  AD 03 C0  |  LDA $C003
E39A  85 FE     |  STA $FE
E39C  AD 04 C0  |  LDA $C004
E39F  85 FF     |  STA $FF
E3A1  8C 10 03  |  STY $0310
E3A4  AD 80 04  |  LDA $0480
E3A7  09 01     |  ORA #$01
E3A9  8D 14 03  |  STA $0314
E3AC  8D 80 04  |  STA $0480
E3AF  A0 00     |  LDY #$00
E3B1  60        |  RTS

;******************************************************************************
; restore track 0 (preserving status of previous command)
; heart of the bug is here...
; command should be 0 (no load head flag)
; this way, the command wouldn't fail when no disk is in drive
;
E3B2  AD FE 04  |  LDA $04FE
E3B5  A2 08     |  LDX #$08
E3B7  20 1C E2  |  JSR $E21C
E3BA  8D FE 04  |  STA $04FE
E3BD  60        |  RTS
E3BE  EA        |  NOP
E3BF  EA        |  NOP


;******************************************************************************
;       IRQ : handler
;******************************************************************************

E3C0  48        |  PHA
E3C1  AD 14 03  |  LDA $0314
E3C4  30 19     |  BMI $E3DF      ; checks if the IRQ comes from disk
E3C6  68        |  PLA            ; ...yes, continue here
E3C7  AD 80 04  |  LDA $0480
E3CA  29 FE     |  AND #$FE
E3CC  8D 80 04  |  STA $0480
E3CF  8D 14 03  |  STA $0314
E3D2  68        |  PLA            ; get rid of the IRQ context !!
E3D3  68        |  PLA
E3D4  68        |  PLA            ; so, we are now in the interrupted routine !
E3D5  AD 10 03  |  LDA $0310
E3D8  29 5D     |  AND #$5D
E3DA  8D FE 04  |  STA $04FE      ; store FDC's status (only interesting flags)
E3DD  58        |  CLI            ; enable interrupts
E3DE  60        |  RTS            ; and return to the *caller* of the interrupted
                                  ; routine (not the interrupted routine itself !)
E3DF  68        |  PLA            ; IRQ doesn't come from disk,
E3E0  4C E6 E0  |  JMP $E0E6      ; go to the normal IRQ handler

;******************************************************************************
; disables timer1 interrupts
```

```
E3E3  48        |  PHA
E3E4  A9 40     |  LDA #$40
E3E6  8D 0E 03  |  STA $030E
E3E9  68        |  PLA
E3EA  60        |  RTS

; enables timer1 interrupts
E3EB  48        |  PHA
E3EC  A9 C0     |  LDA #$C0
E3EE  8D 0E 03  |  STA $030E
E3F1  68        |  PLA
E3F2  60        |  RTS

;***************************************************************************
E3F3  04 24 44 64                    ; drive numbers

;***************************************************************************
;  interpreter routine to load a program... not used
;
E3F7  20 06 E0  |  JSR $E006
E3FA  20 4B E0  |  JSR $E04B
E3FD  20 45 EB  |  JSR $EB45
E400  20 00 E0  |  JSR $E000
E403  88        |  DEY
E404  C8        |  INY
E405  20 60 E0  |  JSR $E060      ; reads a non-blank char
E408  20 00 00  |  JSR $0000      ; incomplete !!
E40B  F0 55     |  BEQ $E462      ; end of command ?
E40D  C9 2C     |  CMP #$2C       ; is it a ',' ?
E40F  D0 15     |  BNE $E426
E411  C8        |  INY            ; yes, reads next char
E412  B1 E9     |  LDA ($E9),Y
E414  C9 4E     |  CMP #$4E       ; is it a 'N' ?
E416  D0 05     |  BNE $E41D
E418  8D 4F C1  |  STA $C14F
E41B  10 E7     |  BPL $E404
E41D  C9 44     |  CMP #$44       ; is it a 'D' ?
E41F  D0 0A     |  BNE $E42B
E421  8D 50 C1  |  STA $C150
E424  10 DE     |  BPL $E404
E426  A2 01     |  LDX #$01       ; invalid command end
E428  4C 1B E0  |  JMP $E01B
E42B  C9 4A     |  CMP #$4A       ; is it a 'J' ?
E42D  D0 15     |  BNE $E444
E42F  8D 41 C1  |  STA $C141      ; yes: Join
E432  A5 9C     |  LDA $9C
E434  38        |  SEC
E435  E9 02     |  SBC #$02
E437  8D 4D C1  |  STA $C14D
E43A  A5 9D     |  LDA $9D
E43C  E9 00     |  SBC #$00
E43E  8D 4E C1  |  STA $C14E
E441  4C 04 E4  |  JMP $E404
E444  C9 41     |  CMP #$41       ; is it a 'A' ?
E446  D0 DE     |  BNE $E426
E448  8D 4F C1  |  STA $C14F
E44B  8D 41 C1  |  STA $C141
E44E  C8        |  INY
```

```
E44F  20 5D E0  |  JSR $E05D     ; reads a number
E452  90 D2     |  BCC $E426
E454  AD 46 C1  |  LDA $C146
E457  8D 4E C1  |  STA $C14E
E45A  AD 45 C1  |  LDA $C145
E45D  8D 4D C1  |  STA $C14D
E460  B0 A3     |  BCS $E405
                                 ; execs the command, ie loads specified file
E462  98        |  TYA
E463  48        |  PHA
E464  20 09 E0  |  JSR $E009     ; loads file
E467  68        |  PLA
E468  A8        |  TAY
E469  AD 4C C1  |  LDA $C14C
E46C  F0 08     |  BEQ $E476
E46E  AD 4F C1  |  LDA $C14F
E471  10 03     |  BPL $E476
E473  6C 4B C1  |  JMP ($C14B)   ; auto-run

E476  AD 4B C1  |  LDA $C14B
E479  D0 03     |  BNE $E47E
E47B  4C 69 E0  |  JMP $E069     ; uncomplete ! (points to 0000)
E47E  C9 03     |  CMP #$03
E480  B0 F9     |  BCS $E47B
E482  20 5A E0  |  JSR $E05A     ; links Basic program lines
E485  6F C5 5F C5
E489  A5 92     |  LDA $92
E48B  85 9D     |  STA $9D
E48D  18        |  CLC
E48E  A5 91     |  LDA $91
E490  69 02     |  ADC #$02
E492  85 9C     |  STA $9C
E494  90 02     |  BCC $E498
E496  E6 9D     |  INC $9D
E498  85 9E     |  STA $9E
E49A  85 A0     |  STA $A0
E49C  A5 9D     |  LDA $9D
E49E  85 9F     |  STA $9F
E4A0  85 A1     |  STA $A1
E4A2  A5 A6     |  LDA $A6
E4A4  85 A2     |  STA $A2
E4A6  A5 A7     |  LDA $A7
E4A8  85 A3     |  STA $A3
E4AA  20 5A E0  |  JSR $E05A     ; Basic's RESTORE command
E4AD  1F C9 52 C9
E4B1  20 5A E0  |  JSR $E05A     ; let's the interpreter points to the basic
                            program
E4B4  65 C7 3A C7
E4B8  AD 4B C1  |  LDA $C14B
E4BB  C9 01     |  CMP #$01
E4BD  F0 08     |  BEQ $E4C7
E4BF  2C 4F C1  |  BIT $C14F
E4C2  10 03     |  BPL $E4C7
E4C4  4C 69 E0  |  JMP $E069     ; uncomplete, points to 0000
E4C7  20 5A E0  |  JSR $E05A     ; runs Basic interpreter
E4CA  B5 C4 A8 C4

;*****************************************************************************
```

```
; loads a file

E4CE  AD 2B C1  |  LDA $C12B
E4D1  8D 00 C0  |  STA $C000
E4D4  20 2A E0  |  JSR $E02A     ; checks drive number
E4D7  20 0C E0  |  JSR $E00C     ; searches the file
E4DA  E0 00     |  CPX #$00
E4DC  D0 05     |  BNE $E4E3
E4DE  A2 00     |  LDX #$00      ; File not found
E4E0  4C 1B E0  |  JMP $E01B
E4E3  BD 2F C0  |  LDA $C02F,X   ; File found, reads first sector of it
E4E6  8D 01 C0  |  STA $C001
E4E9  BD 2E C0  |  LDA $C02E,X
E4EC  20 3F E8  |  JSR $E83F
E4EF  A2 00     |  LDX #$00
E4F1  A0 02     |  LDY #$02
E4F3  10 02     |  BPL $E4F7
E4F5  8A        |  TXA
E4F6  A8        |  TAY
E4F7  AD 41 C1  |  LDA $C141
E4FA  D0 0C     |  BNE $E508     ; is it a 'Join' ?
E4FC  B9 25 C0  |  LDA $C025,Y   ; no, uses first start address as global address
E4FF  8D 4D C1  |  STA $C14D
E502  B9 26 C0  |  LDA $C026,Y
E505  8D 4E C1  |  STA $C14E
E508  38        |  SEC           ; computes end address of record
E509  AD 4D C1  |  LDA $C14D
E50C  F9 25 C0  |  SBC $C025,Y
E50F  99 25 C0  |  STA $C025,Y
E512  AD 4E C1  |  LDA $C14E
E515  F9 26 C0  |  SBC $C026,Y
E518  99 26 C0  |  STA $C026,Y
E51B  18        |  CLC
E51C  B9 25 C0  |  LDA $C025,Y
E51F  79 27 C0  |  ADC $C027,Y
E522  99 27 C0  |  STA $C027,Y
E525  B9 26 C0  |  LDA $C026,Y
E528  79 28 C0  |  ADC $C028,Y
E52B  99 28 C0  |  STA $C028,Y
E52E  E0 00     |  CPX #$00
E530  D0 0C     |  BNE $E53E
E532  B9 2A C0  |  LDA $C02A,Y
E535  8D 4C C1  |  STA $C14C
E538  B9 29 C0  |  LDA $C029,Y
E53B  8D 4B C1  |  STA $C14B
E53E  AD 50 C1  |  LDA $C150
E541  30 36     |  BMI $E579     ; is trace required ?
E543  AD 4E C1  |  LDA $C14E     ; if yes, prints addresses
E546  20 4D E1  |  JSR $E14D
E549  AD 4D C1  |  LDA $C14D
E54C  20 4D E1  |  JSR $E14D
E54F  A9 20     |  LDA #$20
E551  20 57 E0  |  JSR $E057
E554  B9 28 C0  |  LDA $C028,Y
E557  20 4D E1  |  JSR $E14D
E55A  B9 27 C0  |  LDA $C027,Y
E55D  20 4D E1  |  JSR $E14D
E560  AD 41 C1  |  LDA $C141
```

```
E563  D0 11     |  BNE $E576
E565  A9 20     |  LDA #$20
E567  20 57 E0  |  JSR $E057
E56A  B9 2A C0  |  LDA $C02A,Y
E56D  20 4D E1  |  JSR $E14D
E570  B9 29 C0  |  LDA $C029,Y
E573  20 4D E1  |  JSR $E14D
E576  20 54 E0  |  JSR $E054
E579  AD 4D C1  |  LDA $C14D
E57C  85 0C     |  STA $0C
E57E  AD 4E C1  |  LDA $C14E
E581  85 0D     |  STA $0D
E583  18        |  CLC
E584  98        |  TYA
E585  69 08     |  ADC #$08
E587  AA        |  TAX
E588  F0 25     |  BEQ $E5AF
E58A  BD 23 C0  |  LDA $C023,X    ; taille du record
E58D  F0 1D     |  BEQ $E5AC
E58F  C9 FF     |  CMP #$FF
E591  D0 03     |  BNE $E596
E593  4C F5 E4  |  JMP $E4F5
E596  8D 41 C1  |  STA $C141
E599  A0 00     |  LDY #$00
E59B  E8        |  INX
E59C  BD 23 C0  |  LDA $C023,X
E59F  91 0C     |  STA ($0C),Y
E5A1  E6 0C     |  INC $0C
E5A3  D0 02     |  BNE $E5A7
E5A5  E6 0D     |  INC $0D
E5A7  CE 41 C1  |  DEC $C141
E5AA  D0 EF     |  BNE $E59B
E5AC  E8        |  INX
E5AD  D0 DB     |  BNE $E58A
E5AF  AD 23 C0  |  LDA $C023
E5B2  8D 01 C0  |  STA $C001
E5B5  AD 24 C0  |  LDA $C024
E5B8  F0 07     |  BEQ $E5C1
E5BA  20 3F E8  |  JSR $E83F
E5BD  A2 02     |  LDX #$02
E5BF  10 C9     |  BPL $E58A
E5C1  60        |  RTS

;*******************************************************************************
; initializes some parameters
E5C2  A9 FF     |  LDA #$FF
E5C4  8D 4F C1  |  STA $C14F
E5C7  8D 50 C1  |  STA $C150
E5CA  8D 3C C1  |  STA $C13C
E5CD  A9 00     |  LDA #$00
E5CF  8D 4D C1  |  STA $C14D
E5D2  8D 4E C1  |  STA $C14E
E5D5  8D 41 C1  |  STA $C141
E5D8  60        |  RTS

E5D9: 46 69 6C 65 20 6E 6F 74 20 66 6F 75 6E 64 00
                File not found.
E5E8: 49 6E 76 61 6C 69 64 20 63 6F 6D 6D 61 6E 64 20 65 6E 64 00
```

```
                Invalid command end.
E5FC: 4E 6F 20 64 72 69 76 65 20 6E 75 6D 62 65 72 00
                No drive number.
E60C: 42 61 64 20 64 72 69 76 65 20 6E 75 6D 62 65 72 00
                Bad drive number.
E61D: 49 6E 76 61 6C 69 64 20 66 69 6C 65 6E 61 6D 65 00
                Invalid filename.
E62E: 44 69 73 63 20 65 72 72 6F 72 00
                Disc error.
E639: 49 6C 6C 65 67 61 6C 20 61 74 74 72 69 62 75 00
                Illegal attribute
E64B: 57 69 6C 64 63 61 72 64 28 73 29 20 6E 6F 74 20 61 6C 6C 6F 77 65 64 00
                Wildcard(s) not allowed
E663: 46 69 6C 65 20 61 6C 72 65 61 64 79 20 65 78 69 73 74 73 00
                File already exists
E677: 49 6E 73 75 66 66 69 63 69 65 6E 74 20 64 69 73 6B 20 73 70 61 63 65 00
                Insufficient disk space
E68F: 53 74 61 72 74 20 61 64 64 72 65 73 73 20 6D 69 73 73 69 6E 67 00
                Start address missing
E6A5: 49 6C 6C 65 67 61 6C 20 71 75 61 6E 74 69 74 79 00
                Illegal quantity
E6B6: 45 6E 64 20 61 64 64 72 65 73 73 20 6D 69 73 73 69 6E 67 00
                End address missing
E6CA: 53 74 61 72 74 20 61 64 64 72 65 73 73 20 3E 20 65 6E 64 20 61 64 64 72 65
                73 73 00 Start address > end address
E6E6: 4D 69 73 73 69 6E 67 20 27 54 4F 27 00
                Missing 'TO'
E6F3: 52 65 6E 61 6D 65 64 20 66 69 6C 65 20 6E 6F 74 20 6F 6E 20 73 61 6D 65 20
                64 69 73 6B 00 Renamed file not on same disk
E711: 4D 69 73 73 69 6E 67 20 63 6F 6D 6D 61 00
                Missing comma
E71F: 53 6F 75 72 63 65 20 61 6E 64 20 64 65 73 74 69 6E 61 74 69 6F 6E 20 64 72
                69 76 65 73 20 6D 75 73 74 20 62 65 20 73 61 6D 65 00
                Source and destination drives must be same
E74A: 44 65 73 74 69 6E 61 74 69 6F 6E 20 6E 6F 74 20 73 70 65 63 69 66 69 65 64
                00 Destination not specified
E764: 43 61 6E 6E 6F 74 20 6D 65 72 67 65 20 61 6E 64 20 6F 76 65 72 77 72 69 74
                65 00 Cannot merge and overwrite
E77F: 53 69 6E 67 6C 65 20 64 65 73 74 69 6E 61 74 69 6F 6E 20 66 69 6C 65 20 6E
                6F 74 20 61 6C 6C 6F 77 65 64 00
                Single destination file not allowed
E7A3: 53 79 6E 74 61 78 20 65 72 72 6F 72 00
                Syntax error

; addresses of the messages above (low bytes in the first line) :
E7B0: D9 E8 FC 0C 1D 2E 39 4B 63 77 8F A5 B6 CA E6 F3 11 1F 4A 64 7F A3
E7C6: E5 E5 E5 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E7 E7 E7 E7 E7 E7


;*****************************************************************************
; error routine
;
E7DC  E8        |  INX
E7DD  8E FF 04  |  STX $04FF
E7E0  6C 49 C1  |  JMP ($C149)   ; clearly, this instruction has been added
                                 ; the error routine is below but not used

E7E3  CA        |  DEX
```

```
E7E4  AD FD 04  |  LDA $04FD
E7E7  29 01     |  AND #$01
E7E9  F0 03     |  BEQ $E7EE
E7EB  4C 69 E0  |  JMP $E069     ; uncomplete, points to 0000
E7EE  E0 16     |  CPX #$16
E7F0  B0 15     |  BCS $E807
E7F2  BD B0 E7  |  LDA $E7B0,X
E7F5  85 0C     |  STA $0C
E7F7  BD C6 E7  |  LDA $E7C6,X
E7FA  85 0D     |  STA $0D
E7FC  20 2D E0  |  JSR $E02D     ; prints message
E7FF  A9 3A     |  LDA #$3A
E801  20 57 E0  |  JSR $E057
E804  4C 13 E8  |  JMP $E813

E807  8A        |  TXA           ; prints the error number
E808  20 4D E1  |  JSR $E14D
E80B  AD FE 04  |  LDA $04FE
E80E  F0 03     |  BEQ $E813
E810  20 4D E1  |  JSR $E14D
E813  4C FF E1  |  JMP $E1FF     ; switch to Basic
E816  60        |  RTS

;*****************************************************************************
; writes a sector
E817  20 06 E2  |  JSR $E206
E81A  AD FE 04  |  LDA $04FE
E81D  F0 05     |  BEQ $E824
E81F  A2 05     |  LDX #$05      ; Disc error
E821  4C 1B E0  |  JMP $E01B
E824  60        |  RTS

;*****************************************************************************
; reads a sector
E825  20 0A E2  |  JSR $E20A
E828  4C 1A E8  |  JMP $E81A

;*****************************************************************************
; reads boot sector
E82B  A9 23     |  LDA #$23
E82D  8D 03 C0  |  STA $C003
E830  A9 C0     |  LDA #$C0
E832  8D 04 C0  |  STA $C004
E835  A9 00     |  LDA #$00
E837  8D 01 C0  |  STA $C001
E83A  8D 0A C0  |  STA $C00A
E83D  A9 01     |  LDA #$01
E83F  8D 02 C0  |  STA $C002
E842  20 24 E0  |  JSR $E024
E845  60        |  RTS

;*****************************************************************************
; checks drive number
;
E846  AE 00 C0  |  LDX $C000
E849  BD 13 C0  |  LDA $C013,X
E84C  F0 01     |  BEQ $E84F
E84E  60        |  RTS
```

```
E84F  A2 03      |  LDX #$03      ; bad drive number
E851  4C 1B E0   |  JMP $E01B

;*******************************************************************************
; prints string pointed by ($0C)
;
E854  A0 00      |  LDY #$00
E856  B1 0C      |  LDA ($0C),Y
E858  F0 06      |  BEQ $E860
E85A  20 57 E0   |  JSR $E057
E85D  C8         |  INY
E85E  10 F6      |  BPL $E856
E860  60         |  RTS

;*******************************************************************************
; not used
E861  AD 46 C1   |  LDA $C146
E864  D0 09      |  BNE $E86F
E866  AD 45 C1   |  LDA $C145
E869  30 04      |  BMI $E86F
E86B  C9 04      |  CMP #$04
E86D  30 02      |  BMI $E871
E86F  A9 FF      |  LDA #$FF
E871  60         |  RTS

;*******************************************************************************
; reads system parameters from boot sector
E872  20 27 E0   |  JSR $E027
E875  A2 07      |  LDX #$07
E877  BD 33 C0   |  LDA $C033,X
E87A  9D 23 C1   |  STA $C123,X
E87D  CA         |  DEX
E87E  10 F7      |  BPL $E877
E880  60         |  RTS

;*******************************************************************************
; writes system parameters to boot sector
E881  20 27 E0   |  JSR $E027
E884  A2 07      |  LDX #$07
E886  BD 23 C1   |  LDA $C123,X
E889  9D 33 C0   |  STA $C033,X
E88C  CA         |  DEX
E88D  10 F7      |  BPL $E886
E88F  20 21 E0   |  JSR $E021
E892  60         |  RTS

;*******************************************************************************
; adds a directory entry (no used)
E893  AD 3E C1   |  LDA $C13E
E896  8D 01 C0   |  STA $C001
E899  AD 3D C1   |  LDA $C13D
E89C  20 3F E8   |  JSR $E83F     ; reads sector (C13D) track (C13E)
E89F  A2 00      |  LDX #$00
E8A1  AC 3F C1   |  LDY $C13F
E8A4  BD 2C C1   |  LDA $C12C,X
E8A7  99 23 C0   |  STA $C023,Y
E8AA  C8         |  INY
E8AB  E8         |  INX
```

```
E8AC  E0 10     |  CPX #$10
E8AE  D0 F4     |  BNE $E8A4
E8B0  EE 25 C0  |  INC $C025
E8B3  20 21 E0  |  JSR $E021
E8B6  60        |  RTS

;*****************************************************************************
E8B7  20 3F E0  |  JSR $E03F
E8BA  F0 08     |  BEQ $E8C4
E8BC  EE 29 C1  |  INC $C129
E8BF  D0 03     |  BNE $E8C4
E8C1  EE 2A C1  |  INC $C12A
E8C4  60        |  RTS

;*****************************************************************************
E8C5  AD 23 C1  |  LDA $C123
E8C8  F0 2B     |  BEQ $E8F5
E8CA  8D 02 C0  |  STA $C002
E8CD  AD 24 C1  |  LDA $C124
E8D0  8D 01 C0  |  STA $C001
E8D3  20 24 E0  |  JSR $E024
E8D6  AD 24 C0  |  LDA $C024
E8D9  8D 23 C1  |  STA $C123
E8DC  AD 23 C0  |  LDA $C023
E8DF  8D 24 C1  |  STA $C124
E8E2  38        |  SEC
E8E3  AD 27 C1  |  LDA $C127
E8E6  E9 01     |  SBC #$01
E8E8  8D 27 C1  |  STA $C127
E8EB  AD 28 C1  |  LDA $C128
E8EE  E9 00     |  SBC #$00
E8F0  8D 28 C1  |  STA $C128
E8F3  A9 01     |  LDA #$01
E8F5  60        |  RTS
;*****************************************************************************
; finds a free directory entry
E8F6  20 24 E0  |  JSR $E024
E8F9  AD 25 C0  |  LDA $C025
E8FC  C9 0F     |  CMP #$0F
E8FE  D0 31     |  BNE $E931      ; this directory sector is full ?
E900  AD 24 C0  |  LDA $C024      ; yes
E903  F0 0C     |  BEQ $E911      ; is it the last directory sector ?
E905  8D 02 C0  |  STA $C002      ; no, reads next one
E908  AD 23 C0  |  LDA $C023
E90B  8D 01 C0  |  STA $C001
E90E  4C F6 E8  |  JMP $E8F6
E911  AD 23 C1  |  LDA $C123      ; yes,
E914  F0 39     |  BEQ $E94F
E916  8D 24 C0  |  STA $C024
E919  AD 24 C1  |  LDA $C124
E91C  8D 23 C0  |  STA $C023
E91F  20 21 E0  |  JSR $E021
E922  20 3F E0  |  JSR $E03F
E925  A9 00     |  LDA #$00
E927  AA        |  TAX
E928  9D 23 C0  |  STA $C023,X
E92B  E8        |  INX
E92C  D0 FA     |  BNE $E928
```

```
E92E  20 21 E0  |  JSR $E021
E931  A2 03     |  LDX #$03       ; looks for a free entry
E933  BD 23 C0  |  LDA $C023,X
E936  F0 07     |  BEQ $E93F
E938  8A        |  TXA
E939  18        |  CLC
E93A  69 10     |  ADC #$10
E93C  AA        |  TAX
E93D  D0 F4     |  BNE $E933
E93F  8A        |  TXA            ; and returns it
E940  8D 3F C1  |  STA $C13F
E943  AD 01 C0  |  LDA $C001
E946  8D 3E C1  |  STA $C13E
E949  AD 02 C0  |  LDA $C002
E94C  8D 3D C1  |  STA $C13D
E94F  60        |  RTS
;*****************************************************************************
E950  20 24 E0  |  JSR $E024
E953  AE 3F C1  |  LDX $C13F
E956  D0 28     |  BNE $E980
E958  20 24 E0  |  JSR $E024
E95B  A2 03     |  LDX #$03
E95D  A9 26     |  LDA #$26
E95F  85 0C     |  STA $0C
E961  A9 C0     |  LDA #$C0
E963  85 0D     |  STA $0D
E965  A0 00     |  LDY #$00
E967  B1 0C     |  LDA ($0C),Y
E969  F0 15     |  BEQ $E980
E96B  A0 08     |  LDY #$08
E96D  B9 2C C1  |  LDA $C12C,Y
E970  C9 3F     |  CMP #$3F
E972  F0 04     |  BEQ $E978
E974  D1 0C     |  CMP ($0C),Y
E976  D0 08     |  BNE $E980
E978  88        |  DEY
E979  10 F2     |  BPL $E96D
E97B  8A        |  TXA
E97C  8D 3F C1  |  STA $C13F
E97F  60        |  RTS
;*****************************************************************************
; points to next directory entry
E980  8A        |  TXA
E981  18        |  CLC
E982  69 10     |  ADC #$10
E984  B0 0E     |  BCS $E994      ; need to read next directory sector ?
E986  AA        |  TAX
E987  A5 0C     |  LDA $0C
E989  69 10     |  ADC #$10
E98B  85 0C     |  STA $0C
E98D  90 D6     |  BCC $E965
E98F  E6 0D     |  INC $0D
E991  4C 65 E9  |  JMP $E965
E994  AD 24 C0  |  LDA $C024      ; yes, gets it...
E997  F0 0C     |  BEQ $E9A5
E999  8D 02 C0  |  STA $C002
E99C  AD 23 C0  |  LDA $C023
E99F  8D 01 C0  |  STA $C001
```

```
E9A2  4C 58 E9  |  JMP $E958
E9A5  A2 00     |  LDX #$00
E9A7  60        |  RTS

;*****************************************************************************
; reads boot sector
E9A8  AD 13 C0  |  LDA $C013
E9AB  D0 FA     |  BNE $E9A7
E9AD  8D 00 C0  |  STA $C000
E9B0  A9 13     |  LDA #$13
E9B2  8D 03 C0  |  STA $C003
E9B5  A9 C0     |  LDA #$C0
E9B7  8D 04 C0  |  STA $C004
E9BA  4C 35 E8  |  JMP $E835
;*****************************************************************************
; location intended to store a command (not used, how would you write to an eprom
                           ?)
;
E9BD: 20 20 20 20 20 20 20 20 20 20 20 20 00

;*****************************************************************************
E9CA  A2 0B     |  LDX #$0B
E9CC  A9 20     |  LDA #$20
E9CE  9D BD E9  |  STA $E9BD,X
E9D1  CA        |  DEX
E9D2  10 FA     |  BPL $E9CE
E9D4  20 60 E0  |  JSR $E060
E9D7  20 00 00  |  JSR $0000
E9DA  F0 69     |  BEQ $EA45
E9DC  38        |  SEC
E9DD  E9 30     |  SBC #$30
E9DF  C9 04     |  CMP #$04
E9E1  B0 0F     |  BCS $E9F2
E9E3  C8        |  INY
E9E4  8D 2B C1  |  STA $C12B
E9E7  A2 09     |  LDX #$09
E9E9  A9 20     |  LDA #$20
E9EB  9D 2B C1  |  STA $C12B,X
E9EE  CA        |  DEX
E9EF  D0 FA     |  BNE $E9EB
E9F1  60        |  RTS

E9F2  A5 EA     |  LDA $EA
E9F4  48        |  PHA
E9F5  A5 E9     |  LDA $E9
E9F7  48        |  PHA
E9F8  98        |  TYA
E9F9  18        |  CLC
E9FA  65 E9     |  ADC $E9
E9FC  85 E9     |  STA $E9
E9FE  90 02     |  BCC $EA02
EA00  E6 EA     |  INC $EA
EA02  20 5A E0  |  JSR $E05A     ; evaluates a Basic expression, result on ACC0
EA05  8B CE 17 CF
EA09  24 28     |  BIT $28
EA0B  10 56     |  BPL $EA63     ; is it a string ?
EA0D  20 5A E0  |  JSR $E05A     ; yes, gets it
EA10  15 D7 D0 D7
```

```
EA14  C9 0C     |  CMP #$0C      ; stores the first 12 chars in E9BD
EA16  90 02     |  BCC $EA1A
EA18  A9 0C     |  LDA #$0C
EA1A  A8        |  TAY
EA1B  88        |  DEY
EA1C  30 08     |  BMI $EA26
EA1E  B1 91     |  LDA ($91),Y
EA20  99 BD E9  |  STA $E9BD,Y
EA23  4C 1B EA  |  JMP $EA1B

EA26  A5 E9     |  LDA $E9
EA28  48        |  PHA
EA29  A9 BD     |  LDA #$BD
EA2B  85 E9     |  STA $E9
EA2D  A9 E9     |  LDA #$E9
EA2F  85 EA     |  STA $EA
EA31  C8        |  INY
EA32  20 45 EA  |  JSR $EA45
EA35  68        |  PLA
EA36  85 EA     |  STA $EA
EA38  68        |  PLA
EA39  18        |  CLC
EA3A  85 E9     |  STA $E9
EA3C  E5 EA     |  SBC $EA
EA3E  49 FF     |  EOR #$FF
EA40  A8        |  TAY
EA41  68        |  PLA
EA42  85 EA     |  STA $EA
EA44  60        |  RTS

EA45  AD 0C C0  |  LDA $C00C
EA48  8D 2B C1  |  STA $C12B
EA4B  20 E7 E9  |  JSR $E9E7
EA4E  C8        |  INY
EA4F  B1 E9     |  LDA ($E9),Y
EA51  88        |  DEY
EA52  C9 CD     |  CMP #$CD
EA54  F0 04     |  BEQ $EA5A
EA56  C9 2D     |  CMP #$2D
EA58  D0 13     |  BNE $EA6D
EA5A  B1 E9     |  LDA ($E9),Y
EA5C  38        |  SEC
EA5D  E9 30     |  SBC #$30
EA5F  C9 04     |  CMP #$04
EA61  90 05     |  BCC $EA68
EA63  A2 04     |  LDX #$04      ; invalid filename
EA65  4C 1B E0  |  JMP $E01B
EA68  8D 2B C1  |  STA $C12B
EA6B  C8        |  INY
EA6C  C8        |  INY
EA6D  A2 00     |  LDX #$00
EA6F  A9 06     |  LDA #$06
EA71  20 8C EA  |  JSR $EA8C
EA74  B1 E9     |  LDA ($E9),Y
EA76  C9 2E     |  CMP #$2E
EA78  D0 08     |  BNE $EA82
EA7A  C8        |  INY
EA7B  A2 06     |  LDX #$06
```

```
EA7D  A9 03     |  LDA #$03
EA7F  20 8C EA  |  JSR $EA8C
EA82  20 00 00  |  JSR $0000
EA85  F0 04     |  BEQ $EA8B
EA87  C9 20     |  CMP #$20
EA89  D0 D8     |  BNE $EA63
EA8B  60        |  RTS
EA8C  8D 41 C1  |  STA $C141
EA8F  B1 E9     |  LDA ($E9),Y
EA91  C9 2A     |  CMP #$2A
EA93  F0 16     |  BEQ $EAAB
EA95  C9 3F     |  CMP #$3F
EA97  F0 07     |  BEQ $EAA0
EA99  20 4E E0  |  JSR $E04E
EA9C  C9 00     |  CMP #$00
EA9E  F0 0A     |  BEQ $EAAA
EAA0  9D 2C C1  |  STA $C12C,X
EAA3  E8        |  INX
EAA4  C8        |  INY
EAA5  CE 41 C1  |  DEC $C141
EAA8  D0 E5     |  BNE $EA8F
EAAA  60        |  RTS
EAAB  A9 3F     |  LDA #$3F
EAAD  9D 2C C1  |  STA $C12C,X
EAB0  E8        |  INX
EAB1  CE 41 C1  |  DEC $C141
EAB4  D0 F7     |  BNE $EAAD
EAB6  C8        |  INY
EAB7  60        |  RTS

; limits a char to alphanumeric
EAB8  C9 30     |  CMP #$30
EABA  90 14     |  BCC $EAD0
EABC  C9 3A     |  CMP #$3A
EABE  90 12     |  BCC $EAD2
EAC0  C9 41     |  CMP #$41
EAC2  90 0C     |  BCC $EAD0
EAC4  C9 5B     |  CMP #$5B
EAC6  90 0A     |  BCC $EAD2
EAC8  C9 61     |  CMP #$61
EACA  90 04     |  BCC $EAD0
EACC  C9 7B     |  CMP #$7B
EACE  90 02     |  BCC $EAD2
EAD0  A9 00     |  LDA #$00
EAD2  60        |  RTS

;*****************************************************************************
; reads first directory sector
;
EAD3  20 33 E0  |  JSR $E033
EAD6  AD 26 C1  |  LDA $C126
EAD9  8D 01 C0  |  STA $C001
EADC  AD 25 C1  |  LDA $C125
EADF  8D 02 C0  |  STA $C002
EAE2  A9 00     |  LDA #$00
EAE4  8D 3F C1  |  STA $C13F
EAE7  4C 45 E0  |  JMP $E045
```

```
;*******************************************************************************
EAEA  A2 09      |  LDX #$09
EAEC  AC 3F C1   |  LDY $C13F
EAEF  B9 2C C0   |  LDA $C02C,Y
EAF2  9D 2C C1   |  STA $C12C,X
EAF5  C8         |  INY
EAF6  E8         |  INX
EAF7  E0 10      |  CPX #$10
EAF9  D0 F4      |  BNE $EAEF
EAFB  60         |  RTS
;*******************************************************************************
EAFC  AE 3F C1   |  LDX $C13F
EAFF  A0 06      |  LDY #$06
EB01  BD 23 C0   |  LDA $C023,X
EB04  C9 20      |  CMP #$20
EB06  D0 03      |  BNE $EB0B
EB08  20 57 E0   |  JSR $E057
EB0B  E8         |  INX
EB0C  88         |  DEY
EB0D  D0 F2      |  BNE $EB01
EB0F  AE 3F C1   |  LDX $C13F
EB12  A0 06      |  LDY #$06
EB14  BD 23 C0   |  LDA $C023,X
EB17  C9 20      |  CMP #$20
EB19  F0 03      |  BEQ $EB1E
EB1B  20 57 E0   |  JSR $E057
EB1E  E8         |  INX
EB1F  88         |  DEY
EB20  D0 F2      |  BNE $EB14
EB22  A9 2E      |  LDA #$2E
EB24  20 57 E0   |  JSR $E057
EB27  A0 03      |  LDY #$03
EB29  BD 23 C0   |  LDA $C023,X
EB2C  20 57 E0   |  JSR $E057
EB2F  E8         |  INX
EB30  88         |  DEY
EB31  D0 F6      |  BNE $EB29
EB33  60         |  RTS

;*******************************************************************************
EB34  A5 0C      |  LDA $0C
EB36  8D 47 C1   |  STA $C147
EB39  A5 0D      |  LDA $0D
EB3B  8D 48 C1   |  STA $C148
EB3E  BA         |  TSX
EB3F  E8         |  INX
EB40  E8         |  INX
EB41  8E 40 C1   |  STX $C140
EB44  60         |  RTS

;*******************************************************************************
; checks no '?' wildcard is used
EB45  A2 08      |  LDX #$08
EB47  BD 2C C1   |  LDA $C12C,X
EB4A  C9 3F      |  CMP #$3F
EB4C  F0 2B      |  BEQ $EB79
EB4E  CA         |  DEX
EB4F  10 F6      |  BPL $EB47
```

```
EB51  60        |  RTS

EB52: 43 4F 4D COM

;*******************************************************************************
EB55  20 06 E0  |  JSR $E006
EB58  A0 00     |  LDY #$00
EB5A  98        |  TYA
EB5B  20 48 EA  |  JSR $EA48
EB5E  AD 32 C1  |  LDA $C132
EB61  C9 20     |  CMP #$20
EB63  D0 0B     |  BNE $EB70
EB65  A2 02     |  LDX #$02
EB67  BD 52 EB  |  LDA $EB52,X
EB6A  9D 32 C1  |  STA $C132,X
EB6D  CA        |  DEX
EB6E  10 F7     |  BPL $EB67
EB70  20 45 EB  |  JSR $EB45
EB73  20 00 E0  |  JSR $E000
EB76  4C 62 E4  |  JMP $E462
;*******************************************************************************
EB79  A2 07     |  LDX #$07      ; prints "wildcards not allowed"
EB7B  4C 1B E0  |  JMP $E01B


;*******************************************************************************
;      RESET : initialisation routine
;*******************************************************************************


EB7E  78        |  SEI           ; inits cpu then waits
EB7F  D8        |  CLD
EB80  A2 FF     |  LDX #$FF
EB82  9A        |  TXS
EB83  E8        |  INX
EB84  8A        |  TXA
EB85  A8        |  TAY
EB86  CA        |  DEX
EB87  D0 FD     |  BNE $EB86
EB89  88        |  DEY
EB8A  D0 FA     |  BNE $EB86
EB8C  9D 00 C0  |  STA $C000,X   ; clears some critical pages
EB8F  9D 00 C1  |  STA $C100,X
EB92  95 00     |  STA $00,X
EB94  9D 00 02  |  STA $0200,X
EB97  CA        |  DEX
EB98  D0 F2     |  BNE $EB8C
EB9A  A2 7A     |  LDX #$7A      ; transfers switching routines in page 4
EB9C  BD ED EE  |  LDA $EEED,X
EB9F  9D 80 04  |  STA $0480,X
EBA2  CA        |  DEX
EBA3  10 F7     |  BPL $EB9C
EBA5  20 AE EE  |  JSR $EEAE     ; checks overlay ram
EBA8  A2 0C     |  LDX #$0C      ; copies a routine in BFE0
EBAA  BD 68 EF  |  LDA $EF68,X   ; to read rom location C002
EBAD  9D E0 BF  |  STA $BFE0,X
EBB0  CA        |  DEX
EBB1  10 F7     |  BPL $EBAA
```

```
EBB3  20 E0 BF  |  JSR $BFE0
EBB6  C0 EA     |  CPY #$EA
EBB8  F0 0F     |  BEQ $EBC9      ; is it a Basic v1.0 ?
EBBA  A9 01     |  LDA #$01
EBBC  8D 07 C0  |  STA $C007      ; indicates a Basic v1.1
EBBF  A9 44     |  LDA #$44
EBC1  8D DC 04  |  STA $04DC
EBC4  A9 47     |  LDA #$47
EBC6  8D E4 04  |  STA $04E4
EBC9  A2 FF     |  LDX #$FF       ; fakes the Basic's initialization
EBCB  86 A9     |  STX $A9
EBCD  A9 FF     |  LDA #$FF
EBCF  A0 97     |  LDY #$97
EBD1  85 A6     |  STA $A6
EBD3  84 A7     |  STY $A7
EBD5  8D C1 02  |  STA $02C1
EBD8  8C C2 02  |  STY $02C2
EBDB  85 A2     |  STA $A2
EBDD  84 A3     |  STY $A3
EBDF  A2 1C     |  LDX #$1C
EBE1  BD CF EE  |  LDA $EECF,X
EBE4  95 E1     |  STA $E1,X
EBE6  CA        |  DEX
EBE7  D0 F8     |  BNE $EBE1
EBE9  AD 07 C0  |  LDA $C007
EBEC  F0 28     |  BEQ $EC16
EBEE  A9 B9     |  LDA #$B9       ; atmos part
EBF0  85 F0     |  STA $F0
EBF2  A9 EC     |  LDA #$EC
EBF4  85 F1     |  STA $F1
EBF6  A9 20     |  LDA #$20
EBF8  8D 4E 02  |  STA $024E
EBFB  A9 04     |  LDA #$04
EBFD  8D 4F 02  |  STA $024F
EC00  A9 00     |  LDA #$00
EC02  8D 60 02  |  STA $0260
EC05  A2 12     |  LDX #$12
EC07  BD 5D EE  |  LDA $EE5D,X
EC0A  9D 38 02  |  STA $0238,X
EC0D  CA        |  DEX
EC0E  10 F7     |  BPL $EC07
EC10  A9 B0     |  LDA #$B0
EC12  A0 CC     |  LDY #$CC
EC14  30 19     |  BMI $EC2F
EC16  A9 FF     |  LDA #$FF       ; ORIC1 part
EC18  A0 BF     |  LDY #$BF
EC1A  8D E1 02  |  STA $02E1
EC1D  8C E2 02  |  STY $02E2
EC20  A2 08     |  LDX #$08
EC22  BD 54 EE  |  LDA $EE54,X
EC25  9D 28 02  |  STA $0228,X
EC28  CA        |  DEX
EC29  10 F7     |  BPL $EC22
EC2B  A9 ED     |  LDA #$ED
EC2D  A0 CB     |  LDY #$CB
EC2F  85 1B     |  STA $1B        ; both
EC31  84 1C     |  STY $1C
EC33  A9 4C     |  LDA #$4C
```

```
EC35  85 1A     |  STA $1A
EC37  85 C3     |  STA $C3
EC39  85 21     |  STA $21
EC3B  8D FB 02  |  STA $02FB
EC3E  A9 A0     |  LDA #$A0
EC40  A0 D2     |  LDY #$D2
EC42  AE 07 C0  |  LDX $C007
EC45  F0 04     |  BEQ $EC4B
EC47  A9 36     |  LDA #$36
EC49  A0 D3     |  LDY #$D3
EC4B  85 22     |  STA $22
EC4D  84 23     |  STY $23
EC4F  8D FC 02  |  STA $02FC
EC52  8C FD 02  |  STY $02FD
EC55  A9 C4     |  LDA #$C4
EC57  A0 04     |  LDY #$04
EC59  8D F5 02  |  STA $02F5
EC5C  8C F6 02  |  STY $02F6
EC5F  A9 00     |  LDA #$00
EC61  8D FF 04  |  STA $04FF
EC64  8D FD 04  |  STA $04FD
EC67  20 5A E0  |  JSR $E05A      ; inits the ORIC with the NMI routine of Basic
EC6A: 88 F8 B8 F8
EC6E  A9 50     |  LDA #$50
EC70  85 31     |  STA $31
EC72  A9 30     |  LDA #$30
EC74  85 32     |  STA $32
EC76  A9 03     |  LDA #$03
EC78  85 C2     |  STA $C2
EC7A  A9 00     |  LDA #$00
EC7C  85 D7     |  STA $D7
EC7E  85 88     |  STA $88
EC80  85 2F     |  STA $2F
EC82  48        |  PHA
EC83  8D 00 05  |  STA $0500
EC86  8D 01 05  |  STA $0501
EC89  8D 02 05  |  STA $0502
EC8C  8D F7 02  |  STA $02F7
EC8F  85 2E     |  STA $2E
EC91  8D F1 02  |  STA $02F1
EC94  8D F2 02  |  STA $02F2
EC97  8D F4 02  |  STA $02F4
EC9A  A9 88     |  LDA #$88
EC9C  85 85     |  STA $85
EC9E  A9 02     |  LDA #$02
ECA0  8D C0 02  |  STA $02C0
ECA3  A9 01     |  LDA #$01
ECA5  A0 05     |  LDY #$05
ECA7  85 9A     |  STA $9A
ECA9  84 9B     |  STY $9B
ECAB  A9 03     |  LDA #$03
ECAD  85 9C     |  STA $9C
ECAF  84 9D     |  STY $9D
ECB1  85 9E     |  STA $9E
ECB3  84 9F     |  STY $9F
ECB5  85 A0     |  STA $A0
ECB7  84 A1     |  STY $A1
ECB9  A2 00     |  LDX #$00      ; prints 'insert system disc'
```

```
ECBB  20 92 EE  |  JSR $EE92
ECBE  A2 09     |  LDX #$09      ; copies SYSTEMDOS filename to C12B
ECC0  BD 40 EE  |  LDA $EE40,X
ECC3  9D 2B C1  |  STA $C12B,X
ECC6  CA        |  DEX
ECC7  10 F7     |  BPL $ECC0
ECC9  A9 8A     |  LDA #$8A      ; initializes Error address to EE8A
ECCB  8D 49 C1  |  STA $C149
ECCE  A9 EE     |  LDA #$EE
ECD0  8D 4A C1  |  STA $C14A
ECD3  A2 D8     |  LDX #$D8      ; 'Force Interrupt' command
ECD5  8E 10 03  |  STX $0310
ECD8  A2 08     |  LDX #$08      ; Restores track 0
ECDA  20 03 E0  |  JSR $E003
ECDD  20 48 E0  |  JSR $E048     ; read sector
ECE0  20 00 E0  |  JSR $E000     ; initializes some parameters
ECE3  20 09 E0  |  JSR $E009     ; loads SYSTEM.DOS
ECE6  20 A3 EE  |  JSR $EEA3     ; clears the top line
ECE9  A2 08     |  LDX #$08      ; copies Basic line "!BOOTUP" to TIB
ECEB  BD 5A ED  |  LDA $ED5A,X
ECEE  95 35     |  STA $35,X
ECF0  CA        |  DEX
ECF1  10 F8     |  BPL $ECEB
ECF3  A2 FF     |  LDX #$FF      ; prints DOS version on top line
ECF5  E8        |  INX
ECF6  BD D0 9F  |  LDA $9FD0,X
ECF9  9D 82 BB  |  STA $BB82,X
ECFC  D0 F7     |  BNE $ECF5

ECFE  A2 1A     |  LDX #$1A      ; copies a routine to BFE0
ED00  BD 3F ED  |  LDA $ED3F,X
ED03  9D E0 BF  |  STA $BFE0,X
ED06  CA        |  DEX
ED07  10 F7     |  BPL $ED00
ED09  A9 AE     |  LDA #$AE      ; prints Basic copyright
ED0B  A0 ED     |  LDY #$ED
ED0D  AE 07 C0  |  LDX $C007
ED10  F0 04     |  BEQ $ED16
ED12  A9 F1     |  LDA #$F1
ED14  A0 ED     |  LDY #$ED
ED16  85 0C     |  STA $0C
ED18  84 0D     |  STY $0D
ED1A  20 2D E0  |  JSR $E02D
ED1D  A2 09     |  LDX #$09      ; copies filename BOOTUPCOM to C12B
ED1F  BD 4A EE  |  LDA $EE4A,X
ED22  9D 2B C1  |  STA $C12B,X
ED25  CA        |  DEX
ED26  10 F7     |  BPL $ED1F
ED28  20 0C E0  |  JSR $E00C     ; searches BOOTUPCOM in directory
ED2B  E0 00     |  CPX #$00
ED2D  D0 0D     |  BNE $ED3C     ; found BOOTUPCOM ? executes !BOOTUP
ED2F  86 35     |  STX $35       ; no, removes !BOOTUP command from TIB
ED31  A9 35     |  LDA #$35
ED33  85 0C     |  STA $0C
ED35  A9 EE     |  LDA #$EE
ED37  85 0D     |  STA $0D
ED39  20 2D E0  |  JSR $E02D     ; and prints Ready
ED3C  4C E0 BF  |  JMP $BFE0     ; goes to ram in order to activate overlay ram
```

```
;*******************************************************************************
; ED3F-ED59 : routine copied to BFE0
;          switches to overlay ram and starts the OS
;
BFE0  78         |  SEI
BFE1  A9 84      |  LDA #$84
BFE3  8D 80 04   |  STA $0480
BFE6  8D 14 03   |  STA $0314
BFE9  20 F8 BF   |  JSR $BFF8
BFEC  A2 34      |  LDX #$34
BFEE  A0 00      |  LDY #$00
BFF0  58         |  CLI
BFF1  20 5A D4   |  JSR $D45A     ; calls the Basic interpreter (no return)
BFF4: CD C4 BD C4
BFF8  6C 4B C1   |  JMP ($C14B)   ; init the OS
;*******************************************************************************

ED5A: 21 42 4F 4F 54 55 50 00 00 !BOOTUP

ED63: 69 6E 73 65 72 74 20 73 79 73 74 65 6D 20 64 69 73 63 00 insert system disc
ED76: 0C 4E 6F 20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74 65 6D 20 6F 6E 20 64
                            69 73 63 08 00 No operating system on disc
ED94: 0C 52 56 31 20 61 64 6A 75 73 74 6D 65 6E 74 20 72 65 71 75 69 72 65 64 08
                            00 RV1 adjustment required
EDAE: 0C 4F 52 49 43 20 45 58 54 45 4E 44 45 44 20 42 41 53 49 43 20 56 31 2E 30
                            0D 0A ORIC EXTENDED BASIC V1.0
EDC9: 60 20 31 39 38 33 20 54 41 4E 47 45 52 49 4E 45 0D 0A 0A 0A ` 1983 TANGERINE
EDDD: 34 37 38 37 30 20 42 59 54 45 53 20 46 52 45 45 0D 0A 0A 00 47870 BYTES FREE
EDF1: 0C 4F 52 49 43 20 45 58 54 45 4E 44 45 44 20 42 41 53 49 43 20 56 31 2E 31
                            0D 0A ORIC EXTENDED BASIC V1.1
EE0C: 60 20 31 39 38 33 20 54 41 4E 47 45 52 49 4E 45 0D 0A 0A 0A ` 1983 TANGERINE
EE20: 20 33 37 36 33 31 20 42 59 54 45 53 20 46 52 45 45 0D 0A 0A 00  37631 BYTES
                            FREE
EE35: 0D 0A 52 65 61 64 79 20 0D 0A 00 Ready

EE40: 00 53 59 53 54 45 4D 44 4F 53 SYSTEMDOS
EE4A: 00 42 4F 4F 54 55 50 43 4F 4D BOOTUPCOM

;*******************************************************************************
; EE54-EE5C: vectors copied to 0228 (ORIC1)
;*******************************************************************************

EE54  4C 03 EC   |  JMP $EC03
EE57  4C 30 F4   |  JMP $F430
EE5A  01 00
EE5C  40         |  RTI

;*******************************************************************************
; EE5D-EE6F: vectors copied to 0238 (atmos)
;*******************************************************************************

EE5D  4C 7C F7   |  JMP $F77C
EE60  4C 78 EB   |  JMP $EB78
EE63  4C C1 F5   |  JMP $F5C1
EE66  4C 65 F8   |  JMP $F865
EE69  4C 22 EE   |  JMP $EE22
EE6C  4C B2 F8   |  JMP $F8B2
EE6F  40         |  RTI
```

```
;******************************************************************************

EE70  A2 31     |  LDX #$31      ; Error: prints 'RV1 adjustment required'
EE72  A0 00     |  LDY #$00      ; and halts the system
EE74  A9 1A     |  LDA #$1A
EE76  99 80 BB  |  STA $BB80,Y
EE79  99 80 BC  |  STA $BC80,Y
EE7C  99 80 BD  |  STA $BD80,Y
EE7F  99 80 BE  |  STA $BE80,Y
EE82  99 FE BE  |  STA $BEFE,Y
EE85  88        |  DEY
EE86  D0 EE     |  BNE $EE76
EE88  F0 02     |  BEQ $EE8C

;*****************************************************************************

EE8A  A2 13     |  LDX #$13      ; Error: prints 'no operating system on disc'
EE8C  20 92 EE  |  JSR $EE92
EE8F  4C 8F EE  |  JMP $EE8F     ; halt the system

;*****************************************************************************

EE92  20 A3 EE  |  JSR $EEA3     ; clears the top line
EE95  4C 9D EE  |  JMP $EE9D     ; and prints a message on it

EE98  E8        |  INX           ; prints a message on the top line
EE99  99 82 BB  |  STA $BB82,Y
EE9C  C8        |  INY
EE9D  BD 63 ED  |  LDA $ED63,X
EEA0  D0 F6     |  BNE $EE98
EEA2  60        |  RTS

EEA3  A0 1B     |  LDY #$1B      ; clears the top line
EEA5  A9 20     |  LDA #$20
EEA7  99 81 BB  |  STA $BB81,Y
EEAA  88        |  DEY
EEAB  D0 FA     |  BNE $EEA7
EEAD  60        |  RTS

;*****************************************************************************

; Checks overlay ram

EEAE  A2 00     |  LDX #$00
EEB0  BD A5 C0  |  LDA $C0A5,X
EEB3  A8        |  TAY
EEB4  A9 55     |  LDA #$55
EEB6  9D A5 C0  |  STA $C0A5,X
EEB9  DD A5 C0  |  CMP $C0A5,X
EEBC  D0 B2     |  BNE $EE70
EEBE  A9 AA     |  LDA #$AA
EEC0  9D A5 C0  |  STA $C0A5,X
EEC3  DD A5 C0  |  CMP $C0A5,X
EEC6  D0 A8     |  BNE $EE70
EEC8  98        |  TYA
EEC9  9D A5 C0  |  STA $C0A5,X
EECC  E8        |  INX
```

```
EECD  D0 E1     |  BNE $EEB0
EECF  60        |  RTS

;***************************************************************************
; EED0-EEE0: interpreter routine copied to E2
;***************************************************************************

00E2  E6 E9     |  INC $E9
00E4  D0 02     |  BNE $00E8
00E6  E6 EA     |  INC $EA
00E8  AD 60 EA  |  LDA $EA60
00EB  C9 20     |  CMP #$20
00ED  F0 F3     |  BEQ $00E0
00EF  20 41 EA  |  JSR $EA41
00F2  60        |  RTS

;***************************************************************************

EEE1:    2C 60 EA 2C 60 EA 60
EEE8: 80 4F C7 52 58

;****************************************************************************
;  EEED-EF67: switching routines transfered to page 4 (address 0480)
;****************************************************************************

0480: 04 00
0482: 00 00                         ; temporary storage for A and flags

0484  4C 60 EA  |  JMP $EA60        ; address replaced for indirect jumps
0487  4C E6 04  |  JMP $04E6        ; enables/disables rom
048A  4C D6 04  |  JMP $04D6
048D  4C DE 04  |  JMP $04DE

0490  08        |  PHP              ; calls a routine in rom or eprom
0491  78        |  SEI              ; destination bank specified in 0481
0492  8D 82 04  |  STA $0482
0495  68        |  PLA
0496  8D 83 04  |  STA $0483
0499  AD 80 04  |  LDA $0480
049C  48        |  PHA
049D  AD 81 04  |  LDA $0481
04A0  20 E6 04  |  JSR $04E6
04A3  AD 83 04  |  LDA $0483
04A6  48        |  PHA
04A7  AD 82 04  |  LDA $0482
04AA  28        |  PLP
04AB  20 84 04  |  JSR $0484
04AE  08        |  PHP
04AF  78        |  SEI
04B0  8D 82 04  |  STA $0482
04B3  68        |  PLA
04B4  8D 83 04  |  STA $0483
04B7  68        |  PLA
04B8  20 E6 04  |  JSR $04E6
04BB  AD 83 04  |  LDA $0483
04BE  48        |  PHA
04BF  AD 82 04  |  LDA $0482
04C2  28        |  PLP
```

```
04C3  60        |  RTS

04C4  A9 00     |  LDA #$00
04C6  8D 81 04  |  STA $0481
04C9  A9 66     |  LDA #$66
04CB  8D 85 04  |  STA $0485
04CE  A9 D4     |  LDA #$D4
04D0  8D 86 04  |  STA $0486
04D3  4C 90 04  |  JMP $0490

04D6  08        |  PHP
04D7  BA        |  TSX
04D8  FE 02 01  |  INC $0102,X
04DB  4C 28 02  |  JMP $0228      ; changed to 0244 for a v1.1

04DE  08        |  PHP
04DF  BA        |  TSX
04E0  FE 02 01  |  INC $0102,X
04E3  4C 2B 02  |  JMP $022B      ; changed to 0247 for a v1.1

04E6  78        |  SEI            ; enables/disables rom
04E7  29 02     |  AND #$02
04E9  8D 81 04  |  STA $0481
04EC  AD 80 04  |  LDA $0480
04EF  29 FD     |  AND #$FD
04F1  0D 81 04  |  ORA $0481
04F4  8D 14 03  |  STA $0314
04F7  8D 80 04  |  STA $0480
04FA  60        |  RTS

;*****************************************************************************
; routine transfered in BFE0, just to read rom location C002...
;*****************************************************************************

EF68  A9 06     |  LDA #$06
EF6A  20 87 04  |  JSR $0487
EF6D  AC 02 C0  |  LDY $C002
EF70  A9 00     |  LDA #$00
EF72  4C 87 04  |  JMP $0487

;*****************************************************************************
;...
;...nothing from EF75 to FFCF
;...
;*****************************************************************************

FFD0: 4F 72 69 63 20 44 4F 53 20 56 30 2E 36 00 00 00 Oric DOS V0.6
FFE0: 28 43 29 20 4F 52 49 43 20 31 39 38 33 00 00 00 (C) ORIC 1983
FFF0: 00 00 00 00 00 00 00 00 00 00
FFFA: AE E0 7E EB C0 E3           ; NMI=E0AE, RESET=EB7E, IRQ=E3C0
```

ANNEXE n° 11
Circuit intégré FD1793
(Contrôleur du lecteur de disquette)

La plus grande partie des informations qui suivent proviennent d'un document en anglais que j'ai récupéré sur le site de Fabrice Francès. Les lecteurs anglophones pourront se référer directement à ce document que je joins à la fin de cette ANNEXE.

Selon Fabrice Broche, les commandes utilisées par la routine XRWTS sont conformes à la notice du 1793 de Western Digital (5"1/4 mini floppy MFM controller/formatter). Cette notice indique qu'il existe 11 commandes qui peuvent être chargées seulement si le bit "Busy Status" est OFF (sauf pour la commande "Force Interrupt"). Le bit "Busy Status" est à 1 pendant l'exécution d'une commande et à 0 après la fin d'exécution. Le "Status Register" indique si l'exécution se termine avec ou sans erreur.

Ces registres servent d'interface entre l'utilisateur et le 1793. La carte contrôleur permet d'y accéder aux adresses suivantes:

310 où se trouve en lecture le "Status Register" et en écriture le "Command Register". C'est là que sont POKÉes les 11 commandes (voir ci-dessous).

311 où se trouve le "Track Register", qui contient le numéro de piste en cours.

312 où se trouve le "Sector Register", qui contient le numéro de secteur en cours.

313 où se trouve le "Data Register", qui contient le numéro de piste désiré ou les data à lire ou à écrire.

Je n'entrerai pas dans les détails de fonctionnement du 1793 qui est assez complexe. Sachez toutefois que les 11 commandes sont divisées en 4 groupes (I à IV). Ces commandes sont de la forme b7 b6 b5 b4 b3 b2 b1 b0. Les b7 à b4 indiquent le numéro de la commande et les b3 à b0 sont les paramètres de la commande. Les valeurs utilisées par SEDORIC sont indiquées.

Groupe I: (déplacement de la tête)

-Restore (#08) positionne la tête sur la piste #00

-Seek (#18) positionne la tête et met à jour le "Track Register"

-Step (#28 ou #38) avance la tête d'une piste dans la même direction que précédemment, et met à jour (#38) ou non (#28) le "Track Register"

-Step in (#48 ou #58) idem vers les n° de pistes croissants

-Step out (#68 ou #78) idem en direction de la piste zéro

Les b0 et b1 (Stepping Motor Rate) des commandes du groupe I indiquent la vitesse de changement de piste. Avec la carte contrôleur MICRODISC (MFM, 1 MHz), ils sont à 1, ce qui correspond à un changement de piste en 30 ms.

Le b2 (Track Number Verify Flag) est à zéro (pas de vérification).

Le b3 (Head Load Flag) est à 1 (Load head at beginning).

Enfin le b4 (Track Update Flag) est soit à 0 (pas de mise à jour du "Track Register"), soit à 1 (mise à jour du "Track Register").

Groupe II: (lecture/écriture d'un ou de plusieurs secteurs)

-Read Sector (#8X) lit un secteur sans tester le n° de face

-Read Sector (#9X) lit plusieurs secteurs sans tester le n° de face

-Write Sector (#AX) écrit un secteur sans tester le n° de face

-Write Sector (#BX) écrit plusieurs secteurs sans tester le n° de face

Avant d'envoyer une commande de lecture/écriture, la tête doit avoir été positionnée sur la bonne piste (commande du groupe I) et donc le "Track Register" doit contenir le n° de piste voulu. L'ordinateur doit encore mettre à jour le "Sector Register" avec le n° du secteur désiré.

Le b0 (Data Address Mark) est toujours à 0 et indique que les data commencent après un #FB.

Le b1 (Side Compare flag) est à 0 s'il ne faut pas et à 1 s'il faut comparer le n° de face lu dans le champ ID sur la disquette et le n° de face indiqué par b3 (voir plus loin).

Le b2 (15 ms Delay) est toujours à zéro (pas de délai).

Le b3 (Side Compare Flag) est à 0 (face n°0) ou à 1 (face n°1). En pratique #80 et #88, par exemple, donnent le même résultat puisque dans les 2 cas le b1 est à 0 (pas de comparaison).

Le b4 (Multiple Record Flag) est à 0 si un seul secteur ou à 1 si plusieurs secteurs doivent être lus ou écrits.

Groupe III:

-Read Address (#C0) le FD1793 lit le prochain champ ID, en assemble les 6 octets (n° de piste, n° de face, n° de secteur, taille du secteur, CRC1 et CRC2), les transfère dans le "Data Register", génère un DRQ pour chaque octet, vérifie la validité et génère une CRC error si besoin.

-Read Track (#E0) tous les octets de gaps, en-têtes et data sont lus, assemblés et transférés dans le "Data Register" et un DRQ est généré pour chaque octet. Il n'y a pas de vérification de CRC, mais le "Lost Data Status Flag" peut éventuellement être mis à 1.

-Write Track (#F0) formate une piste. Toutes les informations à écrire sur la piste doivent être prêtes en mémoire. Il suffit alors de positionner la tête sur la piste à formater, puis d'envoyer la commande #F0. L'écriture commence dès qu'un octet est POKÉ dans le "Data Register" et se continue en suivant des cycles d'horloge.

Attention, tous les octets de #00 à #F4 et #F8 à #FF sont écrits tels quels sur la piste, mais pas les octets de #F5 à #F7. Les #F5 sont convertis en #A1 et le générateur de CRC est initialisé. Les F6 sont convertis en #C2. Finalement, chaque #F7 génère 2 octets de CRC.

Format d'une piste:

Format IBM 34 (256 octets/secteur)		Format ORIC [16 / 17 / 18 / 19] secteurs/piste	
Nombre d'octets	Valeur de l'octet	Nombre d'octets	Valeur de l'octet
80	4E	40 / 40 / 0 / 0	4E
12	00	12 / 12 / 0 / 0	00
3	F6 (écrit C2)	3 / 3 / 0 / 0	F6 (écrit C2)
1	FC (index mark)	1 / 1 / 0 / 0	FC (index mark)
50	4E	40 / 40 / 0 / 0	4E
----- début de cycle d'une piste -------------------		----- début de cycle d'une piste -------------------	
12	00	12	00
3	F5 (écrit A1)	3	F5 (écrit A1)
1	FE (ID)	1	FE (ID)
1	Numéro de Piste	1	Numéro de Piste
1	Numéro de Face	1	Numéro de Face
1	Numéro de Secteur	1	Numéro de Secteur
1	01 (Longueur=256)	1	01 (Longueur=256)
1	F7 (écrit 2 CRC)	1	F7 (écrit 2 CRC)
22	4E	22	4E
12	00	12	00
3	F5 (écrit A1)	3	F5 (écrit A1)
1	FB (Flag Data)	1	FB (Flag Data)
256	Octets de Data	256	Octets de Data
1	F7 (écrit 2 CRC)	1	F7 (écrit 2 CRC)
54 (80?)	4E	40 / 30 / 12 / 12	4E
----- fin de cycle d'une piste ---------------------		----- fin de cycle d'une piste ---------------------	
Jusqu'à la fin de la piste	4E	Jusqu'à la fin de la piste	4E

Note: le début de piste ORIC indiqué ci-dessus soit 96 octets n'est valable que pour 16 ou 17 secteurs/piste. Il est carrément supprimé pour 18 ou 19 secteurs/piste.

Groupe IV:

-Force Interrupt (#DX) commande utilisée pour terminer une commande de lecture/écriture multiple. Les bits b0 à b3 (représentés par un "X") sont positionnés selon diverses "Interupt Conditions". Apparemment

SEDORIC n'utilise pas les commandes #90, #98, #B0, #B8 et #DX. Voir la notice du FD1793 pour toute utilisation spéciale de la routine XRWTS avec ces commandes.

"Status Register"

A réception d'une commande (sauf "Force Interrupt") le b0 est mit à 1 (busy) et les autres bits sont mis à jour en fonction de la nouvelle commande. Si la commande "Force Interrupt" est reçue alors qu'une autre commande est en cours d'exécution, le b0 est mit à 0 et les autres bits sont inchangés. Si la commande "Force Interrupt" est reçue alors qu'aucune autre commande n'est en cours d'exécution, le b0 est mit à 0 et les autres bits sont mis à jour ou à zéro. Après une lecture ou une écriture dans le "Data Register", le bit DRQ (b1) et la ligne DRQ sont mis à zéro.

Résumé du "Status Register"

	Cde de Type I	Lecture Adresse	Lecture Secteur	Lecture Piste	Ecriture Secteur	Ecriture Piste
b7	<-------------------- not ready ------------------->					
b6	Protection Ecriture	<---------- 0 ---------->			<- Protection Ecriture ->	
b5	Head Loaded	0	Record Type	0	<---- Erreur Ecriture --->	
b4	Seek Error	<------------ RNF ----------->		0	<--- RNF -->	0
b3	<----------CRC Error ---------->			0	<-CRC Err->	0
b2	Piste zéro	<--------------- Lost Data --------------->				
b1	Index Pulse	<--------------- DRQ --------------->				
b0	<-------------------- Busy -------------------->					

Status pour commande de type I

b7	"Not Ready"	1 = pas prêt	0= prêt
b6	"Protected"	1 = la disquette est protégée contre l'écriture	
b5	"Head Loaded"	1 = la tête est positionnée	
b4	"Seek Error"	1 = piste désirée pas encore trouvée	0 = "Track Register" mis à jour
b3	"CRC Error"	1 = mauvaise CRC lue dans l'entête du secteur	
b2	"Track 0"	1 = la tête est positionnée sur la piste zéro	
b1	"Index"	1 = #FC ("Index Mark") détecté	
b0	"Busy"	1 = commande en cours	0 = pas de commande en cours

Status pour commande de type II et III

b7	"Not Ready"	1 = pas prêt	0= prêt
b6	"Protected"	1 = la disquette est protégée contre l'écriture	
b5	"Record Type"	1 = #F8 détecté (deleted data addr mark)	0 = #FB détecté (data addr mark)
	"Write Fault"	1 = erreur d'écriture	
b4	"Not Found"	1 = piste, secteur ou face pas trouvée	
b3	"CRC Error"	1 = mauvaise CRC lue dans l'entête du secteur ou dans les data	

b2	"Lost Data"	1 = l'ordinateur n'a pas réagi assez vite au DRQ	
b1	"Data Request"	1 = saturé en lecture ou vide en écriture, reflète la ligne DRQ	
b0	"Busy"	1 = commande en cours	0 = pas de commande en cours

A3. Floppy Disk Controller 1793 brief reference (from Western Digital data sheet)

General description

The FD179X (X=1,2,3,4,5,7) can be considered the end result of both the FD177X and 178X designs. In order to maintain compatibility, the FD177X, FD178X and FD179X were made as close as possible with the instruction set and I/O registers being identical. The 1793 is identical to the 1791 except the Data Access Lines are TRUE (for systems that utilize true data buses). The 1792 and 1794 are "single density only" versions of the 1791 and 1793 respectively. The 1795/7 has a side select output for controlling double sided drives.

Processor interface

The address bits A1 and A0, combined with the signals R/W, are interpreted as selecting the following registers:

A1	A0	Read	Write
0	0	Status Register	Command Register
0	1	Track Register	Track Register
1	0	Sector Register	Sector Register
1	1	Data Register	Data Register

On Disk Read operations, the Data Request is activated when an assembled serial input byte is transferred in paralled to the Data Register. This bit is cleared when the Data Register is read by the processor. If the Data Register is read after one or more character are lost, by having not transferred into the register prior to processor readout, the Lost Data bit is set in the Status Register. The Read operation continues until the end of sector is reached.

On Disk Write operations the Data Request is activated when the Data Register transfers its contents to the Data Shift Register, and requires a new data byte. It is reset when the Data Register is loaded with new data by the processor. If new data is not loaded at the time the next serial byte is required by the floppy disk, a byte of zeroes is written on the diskette and the Lost Data bit is set in the Status Register.

At the completion of every command an INTRQ is generated. INTRQ is reset by either reading the status register or by loading the command register with a new command. In addition, INTRQ is generated if a Force Interrupt command condition is met.

Command description

Command words should only be loaded in the Command Register when the Busy status bit is off (Status bit 0). The one exception is the Force Interrupt command. Whenever a command is being executed, the Busy status bit is set. When a command is completed, an interrupt is generated and the busy status bit is reset. The Status Register indicates whethter the completed command encountered an error or was fault free. For ease of discussion, commands are divided into four types (I, II, III, IV).

COMMAND SUMMARY (models 1791, 1792, 1793, 1794)

Type	Command	b7	b6	b5	b4	b3	b2	b1	b0
I	Restore	0	0	0	0	h	V	r1	r0
I	Seek	0	0	0	1	h	V	r1	r0
I	Step	0	0	1	T	h	V	r1	r0
I	Step-In	0	1	0	T	h	V	r1	r0
I	Step-Out	0	1	1	T	h	V	r1	r0
II	Read Sector	1	0	0	m	S	E	C	0
II	Write Sector	1	0	1	m	S	E	C	a0
III	Read Address	1	1	0	0	0	E	0	0
III	Read Track	1	1	1	0	0	E	0	0
III	Write Track	1	1	1	1	0	E	0	0
IV	Force Interrupt	1	1	0	1	i3	i2	i1	i0

FLAG SUMMARY

r1 r0	Stepping Motor Rate
V	Track Number Verify Flag (0: no verify, 1: verify on dest track)
h	Head Load Flag (1: load head at beginning, 0: unload head)
T	Track Update Flag (0: no update, 1: update Track Register)
a0	Data Address Mark (0: FB, 1: F8 (deleted DAM))
C	Side Compare Flag (0: disable side compare, 1: enable side comp)
E	15 ms delay (0: no 15ms delay, 1: 15 ms delay)
S	Side Compare Flag (0: compare for side 0, 1: compare for side 1)
m	Multiple Record Flag (0: single record, 1: multiple records)
i3 i2 i1 i0	Interrupt Condition Flags i3-i0 = 0 Terminate with no interrupt (INTRQ) i3 = 1 Immediate interrupt, requires a reset i2 = 1 Index pulse i1 = 1 Ready to not ready transition i0 = 1 Not ready to ready transition

Type I commands

The type I commands include the Restore, Seek, Step, Step-In and Step-Out commands. Each of the Type I commands contains a rate field r1 r0 which determines the stepping motor rate.

r1	r0	Stepping rate
0	0	6 ms
0	1	12 ms
1	0	20 ms
1	1	30 ms

An optional verification of head position can be performed by settling bit 2 (V=1) in the command word. The track number from the first encountered ID Field is compared against the contents of the Track Register. If the track numbers compare (and the ID Field CRC is correct) the verify operation is complete and an INTRQ is generated with no errors.

Restore (Seek Track 0)

Upon receipt of this command, the TR00 input is sampled. If TR00 is active (low) indicating the head is positioned over track 0, the Track Register is loaded with zeroes and an interrupt is generated. If TR00 is not active, stepping pulses at a rate specified by the r1 r0 field are issued until the TR00 input is activated. At this time, the Track Register is loaded with zeroes and an interrupt is generated.

Seek

This command assumes that the Track Register contains the track number of the current position of the head and the Data Register contains the desired track number. The FD179X will update the Track Register and issue stepping pulses in the appropriate direction until the contents of the Track Register are equal to the contents of the Data Register. An interrupt is generated at the completion of the command. Note: when using multiple drives, the track register must be updated for the drive selected before seeks are issued.

Step

Upon receipt of this command, the FD179X issues one stepping pulse to the disk drive. The stepping direction motor direction is the same as in the previous step command. An interrupt is generated at the end of the command.

Step-In

Upon receipt of this command, the FD179X issues one stepping pulse in the direction towards track 76. An interrupt is generated at the end of the command.

Step-Out

Upon receipt of this command, the FD179X issues one stepping pulse in the direction towards track 0. An interrupt is generated at the end of the command.

Type II commands

Type II commands are the Read Sector and Write Sector commands. Prior to loading the Type II command into the Command Register, the computer must load the Sector Register with the desired sector number. Upon receipt of the Type II command, the busy status bit is set. The FD179X must find an ID field with a matching Track number and Sector number, otherwise the Record not found status bit is set and the command is terminated with an interrupt. Each of the Type II commands contains an m flag which determines if multiple records (sectors) are to be read or written. If m=0, a single sector is read or written and an interrupt is generated at the completion of the command. If m=1, multiple records are read or written with the sector register internally updated so that an address verification can occur on the next record. The FD179X will continue to read or write multiple records and update the sector register in numerical ascending sequence until the sector register exceeds the number of sectors on the track or until the Force Interrupt command is loaded into the Command Register. The Type II commands for 1791-94 also contain side select compare flags. When C=0 (bit 1), no comparison is made. When C=1, the LSB of the side number is read off the ID Field of the disk and compared with the contents of the S flag.

Read Sector

Upon receipt of the command, the head is loaded, the busy status bit set and when an ID field is encountered that has the correct track number, correct sector number, correct side number, and correct CRC, the data field is presented to the computer. An DRQ is generated each time a byte is transferred to the DR. At the end of the Read operation, the type of Data Address Mark encountered in the data field is recorded in the Status Register (bit 5).

Write Sector

Upon receipt of the command, the head is loaded, the busy status bit set and when an ID field is encountered that has the correct track number, correct sector number, correct side number, and correct CRC, a DRQ is generated. The FD179X counts off 22 bytes (in double density) from the CRC field and the Write Gate output is made active if the DRQ is serviced (ie. the DR has been loaded by the computer). If DRQ has not been serviced, the command is terminated and the Lost Data status bit is set. If the DRQ has been serviced, 12 bytes of zeroes (in double density) are written to the disk, then the Data Address Mark as determined by the a0 field of the command. The FD179X then writes the data field and generates DRQ's to the computer. If the DRQ is not serviced in time for continuous writing the Lost Data Status bit is set and a byte of zeroes is written on the disk (the command is not terminated). After the last data byte has been written on the disk, the two-byte CRC is computed internally and written on the disk followed by one byte of logic ones.

Type III commands:

Read Address

Upon receipt of the Read Address command, the head is loaded and the Busy Status bit is set. The next encountered ID field is then read in from the disk, and the six data bytes of the ID field are assembled and transferred to the DR, and a DRQ is generated for each byte. The six bytes of the ID field are : Track address, Side number, Sector address, Sector Length, CRC1, CRC2. Although the CRC bytes are transferred to the computer, the FD179X checks for validity and the CRC error status bit is set if there is a CRC error. The track address of the ID field is written into the sector register so that a comparison can be made by the user. At the end of the operation, an interrupt is generated and the Busy status bit is reset.

Read Track

Upon receipt of the Read Track command, the head is loaded, and the busy status bit is set. Reading starts with the leading edge of the first encountered index pulse and continues until the next index pulse. All gap, header, and data bytes are assembled and transferred to the data register and DRQ's are generated for each byte. The accumulation of bytes is synchronized to each address mark encountered. An interrupt is generated at the completion of the command. The ID Address Mark, ID field, ID CRC bytes, DAM, Data and Data CRC bytes for each sector will be correct. The gap bytes may be read incorrectly during write-splice time because of synchronization.

Write Track (formatting a track)

Upon receipt of the Write Track command, the head is loaded and the Busy Status bit is set. Writing starts with the leading edge of the first encountered index pulse and continues until the next index pulse, at which time the interrupt is activated. The Data Request is activated immediately upon receiving the command, but writing will not start until after the first byte has been loaded into the DR. If the DR has not been loaded by the time the index pulse is encountered, the operation is terminated making the device Not Busy, the Lost Data status bit is set, and the interrupt is activated. If a byte is not present in the DR when needed, a byte of zeroes is substituted.

This sequence continues from one index mark to the next index mark.

Normally, whatever data pattern appears in the data register is written on the disk with a normal clock pattern. However, if the FD179X detects a data pattern of F5 thru FE in the data register, this is interpreted as data address marks with missing clocks or CRC generation.
The CRC generator is initialized when an F5 data byte is about to be transferred (in MFM). An F7 pattern will generate two CRC bytes. As a consequence, the patterns F5 thru FE must not appear in the gaps, data fiels, or ID fiels.

Tracks may be formatted with sector lengths of 128, 256, 512 or 1024 bytes.

DATA PATTERN	FD179X interpretation in MFM
00 thru F4	Write 00 thru F4
F5	Write A1, preset CRC
F6	Write C2
F7	Generate 2 CRC bytes
F8 thru FF	Write F8 thru FF

IBM system 34 format - 256 bytes/sector

```
Number of Bytes (decimal)          Value of byte written
          80                       4E
          12                       00
           3                       F6 (writes C2)
           1                       FC (index mark)
          50                       4E
        +------
        |  12                      00
        |   3                      F5 (writes A1)
        |   1                      FE (ID address mark)
        |   1                      Track number
        |   1                      Side number
        |   1                      Sector Number
        |   1                      01 (sector length)
        |   1                      F7 (2 CRCs written)
        |  22                      4E
        |  12                      00
        |   3                      F5 (writes A1)
        |   1                      FB (data address mark)
        | 256                      DATA
        |   1                      F7 (2 CRCs written)
        |  54                      4E
        +------
          to the end               4E
```

Type IV command

The Forced Interrupt command is generally used to terminate a multiple sector read or write command or insure Type I status register. This command can be loaded into the command register at any time. If there is a current command under execution (busy status bit set), the command will be terminated and the busy status bit reset.

Status Register

Upon receipt of any command, except the Force Interrupt command, the Busy Status bit is set and the rest of the status bits are updated or cleared for the new command. If the Force Interrupt command is received when there is a current command under execution, the Busy status bit is

reset and the rest of the status bits are unchanged. If the Force Interrupt command is received when there is not a current command under execution, the Busy Status bit is reset and the rest of the status bits are updated or cleared. In this case, Status reflects the Type I commands.

The user has the option of reading the status register through program control or using the DRQ line with DMA or interrupt methods. When the DR is read the DRQ bit in the Status register and the DRQ line are automatically reset. A write to the DR also causes both DRQ's to reset. The busy bit in the status may be monitored with a user program to determine when a command is complete, in lieu of using the INTRQ line. When using the INTRQ, a busy status check is not recommended because a read of the status register to determine the condition of busy will reset the INTRQ line.

STATUS REGISTER SUMMARY

	TYPE I COMMANDS	READ ADDRESS	READ SECTOR	READ TRACK	WRITE SECTOR	WRITE TRACK
b7	not ready	not ready	not ready	not ready	not ready	not ready
b6	wr. protect	0	0	0	wr. prot.	wr. prot.
b5	head loaded	0	record type	0	wr. fault	wr. fault
b4	seek error	RNF	RNF	0	RNF	0
b3	CRC error	CRC error	CRC error	0	CRC error	0
b2	track 0	lost data	lost data	lost data	lost data	lost data
b1	index pulse	DRQ	DRQ	DRQ	DRQ	DRQ
b0	busy	busy	busy	busy	busy	busy

STATUS FOR TYPE I COMMANDS

b7	Not ready	This bit when set indicates the drive is not ready. When reset it indicates the drive is ready. This bit is an inverted copy of the Ready input and logically ORed with MR.
b6	Protected	When set, indicates Write Protect is activated.
b5	Head loaded	When set, it indicates the head is loaded and engaged.
b4	Seek error	When set, the desired track was not verified. This bit is reset to 0 when updated.
b3	CRC error	bad CRC encountered in ID field
b2	Track 00	When set, indicates head is positioned to Track 0.
b1	Index	When set, indicates index mark detected from drive.
b0	Busy	When set, command is in progress. When reset no command is in progress

STATUS FOR TYPE II & III COMMANDS

b7	Not ready	Same as for type I commands
b6	Protected	On Read Record or Read Track, not used. On any write: it indicates a Write Protect. This bit is reset when updated.
b5	Record Type/Write Fault	On Read Record: it indicates the record-type code from data field address mark (1: Deleted Data Mark, 0: Data Mark). On any write: it indicates a Write Fault. This bit is reset when updated.
b4	Record not found	When set, it indicates the desired track, sector, or side were not found. This bit is reset when updated.
b3	CRC error	if b4 is set, an error is found in one or more ID fields

		otherwise it indicates error in data field. This bit is reset when updated.
b2	Lost data	When set, it indicates the computer did not respond to DRQ in one byte time. This bit is reset to zero when updated.
b1	Data ReQuest	This bit is a copy of the DRQ output. When set, it indicates the DR is full on a Read operation or the DR is empty on a Write operation. This bit is reset to zero when updated.
b0	Busy	When set, command is under execution. When reset, no command is under execution.

ANNEXE n° 12
System F.A.Q.

Informations recueillies sur oric@lyghtforce.com
(Contribution de Fabrice Francès)

Voir aussi les articles de Fabrice Broche dans MICR'ORIC et notamment “Bonjour les MICRODISQUES” dans le n°6, pages 35 à 41.

Un grand nombre de questions se posent: comment l'EPROM du MICRODISC est-elle validée (connectée) puis invalidée? Est-il possible de se re-connecter sur cette EPROM après le boot (pour la lire par exemple) et comment? Existe t-il plusieurs versions de cette EPROM du MICRODISC, les EPROM des autres contrôleurs supportant SEDORIC sont-elles identiques ou simplement compatibles. Quel impact cela a t-il sur le fonctionnement de SEDORIC après le boot? Serait-il possible d'avoir dans l'EPROM du MICRODISC un système minimum permettant de formater des disquettes, de lire et écrire des fichiers afin de développer des jeux ou des applications pour lesquels SEDORIC n'est pas forcément indispensable. Ceci permettrait de récupérer la RAM overlay et la page 4 qui ne seraient plus utilisés par SEDORIC. Ces nouvelles applications disposeraient alors de la RAM de 0400 à FFFF (RAM permanente + RAM overlay). Les disquettes seraient alors réduites à jouer le rôle de simple cartouches. Pour ce faire, serait-il possible d'utiliser une EPROM de MICRODISC de 16koctets? Où puis-je trouver des informations?

1) Est-il possible de se re-connecter sur l'EPROM du MICRODISC après le boot (pour la lire par exemple) et comment?

Oui, c'est très simple, il suffit de mettre à zéro le b7 de l'adresse 0314 (0 = EPROM sélectionnée, 1 = EPROM inhibée). Un article de A. Viaud, paru dans THÉORIC n°15, page56, indique comment obtenir le listage de l'EPROM du MICRODISC. Un court programme y est donné. Autre possibilité: les possesseurs d'un programmateur d'EPROM peuvent retirer l'EPROM de la carte contrôleur du MICRODISC, la lire et même la recopier! Les paresseux trouverons en ANNEXE le listing désassemblé provenant du site de Fabrice Francès: http://www.ensica.fr/oric/HARDWARE/Eprom.lst

2) Existe t-il plusieurs versions de cette ROM du MICRODISC? Les ROM des autres contrôleurs supportant SEDORIC sont-elles identiques ou simplement compatibles. Quel impact cela a t-il sur le fonctionnement de SEDORIC après le boot?

La version 1.1 est universellement répandue, cela semble dire qu'il a existé une version 1.0, mais elle n'a peut-être pas été diffusée. En pratique la nature de la ROM importe peut, tant qu'elle est compatible. La ROM du MICRODISC n'a d'ailleurs pas été créée pour SEDORIC, mais pour le DOS V1.1. C'est SEDORIC qui a dû s'adapter et ce n'est d'ailleurs pas très heureux.

3) Serait-il possible d'avoir dans la ROM du MICRODISC un système minimum permettant de formater des disquettes, de lire et écrire des fichiers afin de développer des jeux ou des applications pour lesquels SEDORIC n'est pas forcément indispensable. Ceci permettrait de récupérer la RAM overlay et la page 4 qui ne seraient plus utilisés par SEDORIC. Ces nouvelles applications disposeraient alors de

la RAM de 0400 à FFFF (RAM permanente + RAM overlay). Les disquettes seraient alors réduites à jouer le rôle de simple cartouches.

C'est non seulement possible, mais cela existe déjà. Une routine équivalente à XRWTS existe dans la ROM du MICRODISC en E20C. Il est possible de l'utiliser, mais il faut garder libre la RAM overlay de C000 à C3FF car la ROM du MICRODISC y écrit ses variables et paramètres ainsi que certaines données qu'il lit sur la disquette. De plus, il faut restaurer dans la seconde moitié de la page 4 les routines de "switching" et les variables utilisées par la ROM du MICRODISC. Ces routines et variables sont à recopier de la zone EEED à EF67. Les points d'entrée dans la ROM du MICRODISC sont en E021 pour écrire et en E024 pour lire un secteur.

Alternativement, on peut réduire SEDORIC à la seule routine XRWTS en CFCD. C'est la routine qui gère la lecture et l'écriture sur les disquettes. Le reste de la RAM overlay est alors disponible. Cette méthode a été utilisée pour PINFORIC par José Maria Enguitad et Fabrice Francès et semble meilleure que d'utiliser les routines de la ROM du MICRODISC en jouant avec les bits de l'adresse 0314, car dans ce dernier cas, les applications développées ne tournerons pas sur le TELESTRAT En effet, ce dernier sélectionne la RAM overlay d'une manière différente et le problème est déjà corrigé dans la cartouche STRATORIC.

4) Pour ce faire, serait-il possible d'utiliser une ROM de MICRODISC de 16koctets? Fabrice Francès a crée une ROM de 16Koctets pour la carte du MICRODISC. Cette ROM permet d'émuler le TELESTRAT sur un ATMOS.

4) Où puis-je trouver des informations? Principalement sur le site de Fabrice Francès, notamment dans les pages:

http://www.ensica.fr/oric/hardware_english.html
http://www.ensica.fr/oric/hardware_francais.html
http://www.ensica.fr/oric/emulate_english.html
http://www.ensica.fr/oric/archive_english.html
http://www.ensica.fr/oric/HARDWARE/Eprom.lst

Mais aussi:

http://www.algonet.se/~hakan_k/index.html (manuel de l’ORIC-1)
http://rrzs42.uni-regensburg.de/~hep09515/oric.html
(schémas ORIC-1 et MICRODISC)

ANNEXE n° 13
Exercices de passage ROM <--> RAM overlay

En RAM overlay, l'adresse C024 (ATMORI) contient la valeur #00 pour les machines équipées d'une ROM V1.0 et la valeur #80 pour celles qui sont équipées de la V1.1. En ROM, cette adresse contient la valeur #31 (ROM V1.0) ou la valeur #08 (ROM V1.1).

Allumez votre machine et tapez: ?PEEK(#C024), l'écran affiche alors 8 si ROM V1.1 ou 49 si ROM V1.0. Normalement, vous êtes donc sur la ROM.

L'accès à la RAM overlay qui contient SEDORIC est prévu (heureusement!). Le "truc" est simple et indiqué dans le manuel SEDORIC (ANNEXE 8: passages RAM <-> ROM). Il suffit, dans le programme en langage machine de faire un JSR 04F2 pour accéder à la RAM overlay, d'appeler le ou les sous-programmes voulus et de terminer par un autre JSR 04F2 pour revenir aux conditions normales, c'est à dire sur la ROM. Cette procédure n'affecte aucun registre (ou plutôt, ils sont sauvés puis restaurés).

Voici donc un petit exercice, tapez le programme qui suit. Il s'agit d'un chargeur de langage machine écrit en BASIC:

```
100 DATA #20, #F2, #04          :REM JSR 04F2
110 DATA #AD, #24, #C0          :REM LDA C024
120 DATA #8D, #0E, #98          :REM STA 980E
130 DATA #20, #F2, #04          :REM JSR 04F2
140 DATA #60, #EA               :REM RTS et NOP où sera mis le résultat
200 FOR K=#9801 TO #980E        :REM on place ce programme de 9801 à 980E
210 READ V:POKE K,V
220 NEXT
```

Maintenant sauvegardez votre programme, puis faites un CALL#9801 suivi d'un ?PEEK(#980E) qui vous affichera 128 (soit #80) si votre ROM est une V1.1 ou 0 si c'est une version 1.0

Il faut encore noter que lorsqu'on exécute une commande SEDORIC, on est sous RAM overlay. Si cette commande concerne des manipulations de la mémoire, il est donc possible de consulter, voire d'altérer SEDORIC lui-même. Ainsi pour modifier SEDORIC, vous avez le choix entre 2 méthodes.

Vous pouvez travailler directement sur la disquette en utilisant un éditeur de secteurs du type BDDISK ou NIBBLE et en vous aidant des tableaux "Emplacement de SEDORIC sur une disquette master" en ANNEXE pour avoir la correspondance entre l'adresse en RAM overlay et les coordonnées piste/secteur.

Vous pouvez aussi plus simplement effectuer un SAVE de la zone à modifier, suivi d'un LOAD,A en mémoire basse, modifier cette zone à l'aide d'un moniteur/assembleur/désassembleur classique, la resauver (utilisez les adresses en mémoire basse), la recharger en RAM overlay à l'aide d'un LOAD,A et enfin pérenniser votre travail à l'aide d'un INIT qui recopiera sur disquette le code présent en RAM overlay. La deuxième méthode est un peu plus longue que la première, mais beaucoup plus aisée et confortable.

Voici la liste des logiciels "moniteur/assembleur/désassembleur" qui ont été adaptés pour fonctionner avec SEDORIC, que j'ai testés et qui peuvent être obtenus au CEO (avec entre parenthèses l'auteur de l'adaptation, qu'ils en soient remerciés, mes excuses pour les oublis et omissions):

- Automon de André Chénière (D.Henninot),
- Hades de ERE Informatique (D.Henninot),
- Sédutil de F.Taraud (D.Henninot),
- Supmon et Supdes de J.P.Laurent (semble être le code originel) et
- Assembleur de Micrologic (François Launay).

Il serait pratique de créer une BANQUE n°8 contenant un utilitaire de ce type.

ANNEXE n° 14
Utilisation d'une commande SEDORIC sans argument à partir d'un programme écrit en langage machine

Vous pouvez utiliser la routine 04F2 comme indiqué à l'ANNEXE 10.

Par exemple, pour exécuter la commande SEDORIC OLD il suffit d'insérer dans votre programme "Langage Machine" la séquence: JSR 04F2 JSR E0AF JSR 04F2, simple non?

Tapez le petit programme qui suit:

```
100 DATA #20, #F2, #04          :REM JSR 04F2
110 DATA #20, #AF, #E0          :REM JSR E0AF
120 DATA #20, #F2, #04          :REM JSR 04F2
130 DATA #60                    :REM RTS
200 FOR K=#9801 TO #980A        :REM on place ce programme
210 READ V:POKE K,V             :REM de 9801 à 980A
220 NEXT
```

Sauvegardez, implantez avec un RUN, effacez le programme avec un NEW et restaurez le avec un CALL #9801 Un LIST vous persuadera que ça marche!

Si vous préférez utiliser un moniteur, par exemple Supmon, pas de problème, charger ce moniteur, tapez:

```
I 9801 20 F2 04 20 AF E0 20 F2 04 60 <RETURN>
F <RETURN>
10 REM ESSAI OLD <RETURN>
NEW
CALL #9801
LIST
```

ANNEXE n° 15
Utilisation d'une routine en RAM overlay à partir d'un programme écrit en langage machine

Tapez le petit programme qui suit:

```
100 DATA #48, #45, #4C, #4C, #4F, #20      :REM message "HELLO_
110 DATA #41, #4F, #44, #52, #45, #00      :REM ANDRE" terminé par zéro
120 DATA #20, #F2, #04                     :REM JSR #04F2
130 DATA #A9, #01                          :REM LDA #01
140 DATA #A0, #98                          :REM LDY #98
150 DATA #20, #37, #D6                     :REM JSR D637
160 DATA #20, #F2, #04                     :REM JSR #04F2
170 DATA #60                               :REM RTS
200 FOR K=#9801 TO #981A                   :REM on place ce programme
210 READ V:POKE K,V                        :REM de #9801 à #981A
220 NEXT
```

Sauvegardez ce chef d'oeuvre, RUN pour implanter, CALL #980D pour afficher "HELLO ANDRE". Vous avez utilisé le sous-programme XAFSTR situé en D637 en RAM overlay. Ce sous-programme permet d'afficher toute chaîne d'adresse AY terminée par un zéro. Il y a encore des centaines de routines en RAM overlay qui ne demandent qu'à être utilisées. Faites votre choix à l'aide de "SEDORIC À NU"!

Si vous préférez utiliser un moniteur, par exemple Supmon, tapez:

```
T 9801 "HELLO ANDRE" <RETURN>
I 980C 00 20 F2 04 A9 01 A0 98 20 37 D6 20 F2 04 60 <RETURN>
F <RETURN>
```

CALL #980D affiche le message ou un autre, selon votre fantaisie, mais attention à l'adresse du CALL si la longueur du message est différente.

ANNEXE n° 16
Utilisation d'une commande SEDORIC avec paramètres à partir d'un programme écrit en langage machine

Mais, diriez-vous comment faire avec une commande comportant des paramètres? Un embryon de solution est indiqué dans la BANQUE zéro, lorsque SEDORIC exécute les instructions de démarrage (INIST): il suffit de placer la ou les commandes SEDORIC avec paramètres dans le TIB (tampon clavier), d'initialiser correctement TXTPTR et de faire appel à l'interpréteur en ROM. Cette méthode, très simple, a un inconvénient: on retourne au Ready.

Copie les instructions terminées par "fin de commandes" dans le TIB

```
100 DATA #44,#49,#52,#22,#2A,#2E,#42,#41,#53,#22,#00    :REM DIR"*.BAS"#00
110 DATA #A2, #0B                                       :REM LDX #0B pour copier 12 octets
120 DATA #BD, #01, #98                  :REM LDA 9801,X lit les octets de 9801 à 980B
130 DATA #95, #35                       :REM STA 35,X et les copie dans le TIB de 35 à 3F
140 DATA #CA                            :REM DEX octet précédent (par la fin)
150 DATA #10, #F8                       :REM BPL 120 et reboucle tant qu'il en reste
```

Exécute les instructions présentes dans le tampon clavier avec l'interpréteur BASIC et retour au "Ready"

```
200 DATA #A2, #34                       :REM LDX #34 XY pour ajuster TXTPTR à 0034
210 DATA #A0, #00                       :REM LDY #00 adresse C4BD de l'interpréteur
220 DATA #20, #BD, #C4                  :JSR C4BD ATMOS (prendre C4CD pour l'ORIC-1)
```

Mise en place du programme

```
300 DATA #4C, #B5, #FA                  :REM  JMP  FAB5  SHOOT (FA9B pour la ROM
                                        V1.0)
310 FOR K=#9801 TO #981F:READ V:POKE K,V: NEXT
```

CALL#980C <RETURN>

Affiche le catalogue des fichiers "*.BAS" et retourne au "Ready" sans SHOOT (et oui!).

Une autre méthode, préconisée par Denis Henninot, permet de retourner au point d'appel dans le programme appelant en langage machine. Denis utilise le moniteur Hadès, mais on peut aussi procéder avec un autre assembleur ou avec un chargeur BASIC comme ci-dessus. Sa méthode consiste à détourner le vecteur 1B/1C vers le programme appelant. Sur la ROM, l'affichage du message "Ready" se fait par un JSR

001A avec en 1B/1C l'adresse CCB0 qui est celle de la routine "Afficher la chaîne AY". Cette méthode simple de mise en oeuvre permet de bénéficier de la gestion des erreurs:

I <RETURN> pour inserer le texte source

```
1 ORG $9801
2 CMD NUL 'DIR"*.BAS"'         ;ou autre chaîne à exécuter (avec paramètres)
3 DEB LDX #$00                 ;transfert
4 >1        LDA CMD,X          ;de la (ou des)
5           STA $35,X          ;commande(s)
6           BEQ >2             ;et des éventuels paramètres
7           INX                ;dans le
8           BNE <1             ;buffer clavier (35 à 84)
9 >2        LDA #RET           :LL de l'adresse de retour pour détournement du
10          STA $1B            ;vecteur 1B/1C (affichage du "Ready") vers RET
11          LDA /RET           :idem avec HH
12          STA $1C
13          LDX #$34           ;pour ajuster TXTPTR juste avant le début de commande CMD
14          LDY #$00           ;au début du tampon clavier
15          JMP $C4BD          ;continue à l'Interpréteur (C4CD si ROM V1.0)

16 RET      LDA #$B0
17          STA $1B
18          LDA #$CC           ;CCB0 (CBED si ROM V1.0) affichage du "Ready"
19          STA $1C
20          JSR $FAB5          ;FA9B si ROM V1.0
21          RTS
22          <RETURN>
```

A <RETURN> pour assembler

```
Hadès affiche:     fin des labels $29B8
                   ORG $9801        ;début du sous-programme
                   CMD $9801        ;adresse de la commande
                   DEB $980C        ;adresse d'exécution du sous-programme
                   RET $9827        ;adresse de retour après passage sous SEDORIC
                   FIN $9833        ;fin du sous-programme
```

B <RETURN> pour retourner au BASIC

SAVE"SP",A#9801,E#9833 <RETURN> pour sauver le sous-programme

CALL#980C <RETURN> Affiche le catalogue des fichiers "*.BAS", suivi d'un SHOOT et retour au "Ready"

Pour lire ou écrire un secteur

Voici un autre exemple, fourni par Denis Henninot, qui permet d'utiliser une routine SEDORIC avec paramètres à partir d'un programme en langage machine:

```
ORG $9800
JSR $04F2              ;passage sur la RAM overlay
LDA #$..               ;n° du drive à utiliser (de 0 à 3)
STA $C000              ;que l'on place dans DRIVE
LDA #$..               ;n° de la piste à lire ou à écrire
STA $C001              ;que l'on place dans PISTE
LDA #$..               ;n° du secteur à lire ou à écrire
STA $C002              ;que l'on place dans SECTEUR
LDA #BUFFER            ;LL de l'adresse du tampon lecture/écriture
STA $C003              ;que l'on souhaite utiliser, par exemple 9900
LDA /BUFFER            ;idem HH
STA $C004              ;cette adresse est placée dans RWBUF
```

et l'une des deux instructions suivantes doit être insérée:

```
JSR $DA73              ;XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
JSR $DAA4              ;XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
JSR $04F2              ;et enfin retour sur la ROM
```

Rechercher un secteur disponible sur la disquette

Encore une routine de Denis, très utile pour les amateurs de langage machine qui désire utiliser au mieux les avantages de SEDORIC.

```
JSR $04F2              ;passage sur la RAM overlay
LDA #$..               ;n° du drive à utiliser (de 0 à 3)
STA $C000              ;que l'on place dans DRIVE
JSR $DA4C              ;XPMAP prend le secteur de bitmap dans BUF2
JSR $DC6C              ;XLIBSE cherche un secteur libre, revient avec coordonnées AY
STA $C001              ;que l'on place dans PISTE
STY $C002              ;et dans SECTEUR
JSR $DA8A              ;XSMAP sauve le secteur de bitmap sur la disquette
JSR $04F2              ;et enfin retour sur la ROM
```

Libère un secteur déjà occupé

Cette routine, qui est la contrepartie de la routine précédente, est elle aussi bien utile.

```
JSR $04F2                    ;passage sur la RAM overlay
LDA #$..                     ;n° du drive à utiliser (de 0 à 3)
STA $C000                    ;que l'on place dans DRIVE
JSR $DA4C                    ;XPMAP prend le secteur de bitmap dans BUF2
LDA $C001                    ;PISTE du secteur à libérer
LDY $C002                    ;SECTEUR à libérer
JSR $DD15                    ;XDETSE libère le secteur Y de la piste A sur la bitmap
JSR $DA8A                    ;XSMAP sauve le secteur de bitmap sur la disquette
JSR $04F2                    ;et enfin retour sur la ROM
```

ANNEXE n° 17
LES BOGUES DE SEDORIC
(Sans vouloir porter atteinte à ce système d'exploitation génial)

Le lecture de cette ANNEXE démontre que la version 3.0 est bien loin d'être totalement corrigée. Les bogues principales ont été traitées. Pour les autres, il faudra encore du temps et du recul. Eliminer une bogue nécessite non seulement d'en trouver la cause et de mettre au point un traitement, mais aussi et surtout de vérifier que la correction ne sera pas pire que le mal. Le code de SEDORIC est très optimisé et touffu. Il est parfois difficile de se rendre couper de toutes les implications qu'une modification peut entraîner.

<u>Problème de l'utilisation des minuscules</u>

Ce problème est totalement ***corrigé*** : l'utilisation des minuscules n'est désormais plus supportée dans les commandes SEDORIC. Vous trouverez ci-dessous la liste des inconvénients que cela apportait.

Le manuel indique (page 22) qu'il est possible de taper les commandes SEDORIC en minuscules. Ceci n'est pas très pratique, puisqu'il faut continuer à entrer les commandes BASIC en MAJUSCULES, ce qui entraîne une continuelle utilisation du CTRL/T. Enfin, il y a de nombreux problèmes avec l'utilisation des minuscules pour entrer les commandes SEDORIC: ça ne marche pas à tous les coups (nombreuses bogues).

Le nom des commandes SEDORIC contenant un token BASIC a été bien géré (voir la table des mots-clés en C9DE/CBBA). Mais certaines commandes SEDORIC exigent en outre un token BASIC pour satisfaire leur syntaxe et là, rien n'a été prévu. Cela ne prête pas à conséquence lorsqu'il s'agit de "-" (commande DELETE ou lecteur-), "<" (commandes RSET et LSET), ">" (commande NC > variable), "&" (commandes NL et -NL), "@" (commande LINPUT), ? et PRINT (commande EXT) mais c'est catastrophique pour: "AUTO" (commandes SAVE et STATUS), "TO" (commandes REN, BACKUP, COPY, CHANGE et FIELD), END (commande NUM), ELSE, GOTO et THEN (commande KEYIF), NEXT (commande RESUME), LPRINT (commande WIDTH), DEF (commande USER) et enfin la commande RESTORE.

La présence des octets correspondant à ces caractères et tokens est demandée à TXTPTR par la routine D22E. Cette routine n'effectue bien sûr aucun contrôle ni aucune conversion. Elle est utilisée aussi pour détecter la présence de certaines options, mais pas de toutes. Par exemple, pour la commande USER, "DEF" et "O" doivent être tapés en MAJUSCULES, par contre "A", "X", "Y" et "P" sont acceptés en minuscule! Ces 4 dernières lettres sont lues par la routine D398 qui lit un caractère à TXTPTR avec conversion en MAJUSCULE.

Voici une liste des options qu'il faut absolument taper en MAJUSCULES: "S" (mais pas "A") pour la commande DKEY, "L" de la commande MERGE, "M" (mais pas "S") pour la commande SEEK, "O" (mais pas "A", "X", "Y" ou "P") de la commande USER, "D" (mais pas "S") de la commande TRACK.

Les options "S", "D" et "R" de la commande OPEN ne posent pas de problème (il faut seulement laisser un espace entre OPEN et D).

Les options qui sont précédées d'une virgule sont correctement traitées, c'est à dire converties en MAJUSCULE. C'est le cas de:

",C" et ",N" des commandes COPY, COPYO et COPYM

",O" de la commande FIELD

",S" et ",D" de la commande INIT

",C", ",E", ",J", ",K" et ",S" pour la commande LINPUT

",A", ",V", ",J" et ",N" des commandes LOAD et chargement direct

",A", ",E" et ",T" des commandes SAVE, SAVEO, SAVEM et SAVEU

",A" et ",T" de la commande STATUS.

La validation du drive indiqué après les commandes: BACKUP, DELBAK, DKEY, DNAME, DNUM, DSYS, DTRACK, INIST, INIT, OPEN D, PMAP, PUT, SMAP, SYSTEM et TAKE se passe sans problème grâce à la routine E60D.

La validation d'un drive spécifié par un nom de fichier ambigu après les commandes: COPY, DEL, DESTROY, DIR, LDIR, PROT, REN, SEARCH et UNPROT se passe sans problème grâce à la routine D451.

Enfin, pour être complet, les commandes “delete” et “using” qui étaient précédemment utilisables en minuscules ne peuvent maintenant être employées qu’en majuscules, car elles ont été remplacées par CHKSUM et VISUHIRES.

<u>Fautes d'orthographe dans les messages affichés:</u>

"UNKNOW'N" (pour "UNKNOWN") en CEAF et CED4 ***(non corrigé)***,

"sectors free" (au lieu de "free sectors") en CF34 ***(non corrigé)***,

"Founds" (pour "Found") en C7B9c ***(non corrigé)***,

"LINES_ALREADY_EXISTS" (au lieu de "LINE_ALREADY_EXISTS") en C7EBc ***(non corrigé)***.

<u>Valeurs incorrectes:</u>

#4E en C531c et en C550c (il faudrait #4F, commande CHANGE, longueur des chaînes) ***(non corrigé)***.

#31 en D71B (il faudrait #32, le n° d'erreur utilisateur minimal est 50 et non 49) ***(non corrigé)***.

#0C en E558 (il faudrait #0B, commande REN, comparaison des "?" de l'ancien nom et du nouveau nom: cette comparaison inclut un octet de trop) ***(non corrigé)***.

#4F en E8A7 (il faudrait #50, commande TKEN, longueur de la chaîne, le manuel indique 79 caractères) ***(non corrigé)***.

#02 en EA1B (il faudrait #03, commande EXT, la validité du troisième caractère de l'extension n'est pas vérifiée. Il est donc possible de mettre n'importe quoi comme troisième caractère. Mais attention quand même, ce n'est pas sans risque: une extension à "CO?" est acceptée, mais les fichiers "*.CO?" ne le seront pas). ***Corrigé*** en C432g (après déplacement de la commande dans la BANQUE n°7).

#5F en F1D3 (il fallait #5E, commande INIT, pour charger les 94 premiers secteurs de la disquette master et non les 95 premiers). Cette valeur a été ***corrigée*** pour tenir compte de la nouvelle BANQUE n°7 et vaut maintenant #63 (99). Par contre la bogue portant sur le nombre de secteurs à charger pour formater une disquette SLAVE est toujours ***non corrigée***.

#63 en FBA4 (il faudrait #3F, c'est à dire 63 en décimal, cette bogue est très grave et empêche absolument l'utilisation de la commande CLOSE sans paramètre) ***(non corrigé)***.

<u>Code incorrect:</u>

040E et 043A bogue CSAVE / CLOAD. L'interférence entre SEDORIC et la ROM a été ***corrigée*** en C60E, C63A, C70E et C73A où l'adresse 0E a été remplacée par l'adresse C1.

C5A4c/C5A6c résidu de mise au point mal digéré dans la commande CHANGE ***(non corrigé)***.

C4D5e Version 2.0: le saut à "ILLEGAL_QUANTITY_ERROR" appelé depuis C4A9 et C4AD ne marche plus car il manque le JMP. ***Corrigé*** dans la version 2.1.

C69Ae Une bogue de la BANQUE n°5 affectait les commandes DKEY, DNAME, DNUM, DSYS, DTRACK, INIST & TRACK . La routine C6DB "Demande la disquette cible" était boguée (mauvaise gestion de "ESC") et a été remplacée par une nouvelle routine en C7A0. L'ancien JSR C6DB a été remplacé pour pouvoir utiliser la routine déboguée. ***Corrigé*** par Ray.

C64Af/C76Bf problème de la mise à jour du flag Double face. La commande INIT était sévèrement boguée. Le paramètre ",D" provoquait bien un formatage en Double face, mais l'indicateur de Double face (le b7 de l'octet n°#09 de la bitmap) n'était pas mis à jour, ainsi qu'en témoignait le directory, qui indiquait désespérément "S/" au lieu de "D/". Cette bogue était très gênante car elle se répercutait sur d'autres commandes, notamment BACKUP. Cette bogue a été ***corrigée*** à partir de la version 2.0 en C64Af et C76Bf.

D16F routine Affiche le message "DISP_TYPE_MISMATCH_ERROR", le LDA #A3 doit être remplacé par un LDX #A3 ***(non corrigé)***.

D479/D47D rempli BUFNOM de "?", or X n'est pas nul en entrée mais vaut #FF (sortie de la boucle D465/D469) donc le premier "?" est écrit en C128 au lieu de C029! De plus au retour Z = 0 car le dernier DEX entraine X = #0B (non nul). Le BEQ suivant ne sert donc à rien ***(non corrigé)***.

D4FD/D505 il y a un JSR D7BD, qui valide le drive demandé, de trop ***(non corrigé)***.

D801/D802 la variable EO est inutilisée et semble être un résidu de mise au point ***(non corrigé)***.

D907/D927 bogue “LOVE” (routine "Prendre un caractère au clavier"): le sous-programme traitant des codes correspondant aux mots-clés SEDORIC était complètement bogué et ne marchait pas. Ceci a été ***corrigé*** en D90A et EA30.

DE80/DE87 il y a un LDY 0269 de trop, il s'agit d'une bogue mineure due à la fatigue du programmeur! ***(non corrigé)***.

E1F8/E20A routine XLOADA, bogue évitée de justesse pour l'option ",V" car Z = 1 par chance, il aurait été mieux en E1F8 de brancher en E20A ***(non corrigé)***.

E38E/E39B commande DIR, bogue bénigne: il aurait fallu un BNE E39B ***(non corrigé)***.

E68C/E6BB commande STATUS, incompatibilité entre les options "T" et "AUTO", absence de vérification: ces options ne doivent pas être utilisées conjointement ***(non corrigé)***.

E6C1/E6CF commande STATUS, absence de vérification du type de fichier avant de forcer le flag AUTO ***(non corrigé)***.

E8CE/E8D5 commande TKEN, il y a ici une bogue potentielle, car au moins un caractère est écrit, même si la longueur de la chaîne set nulle. L'octet lu en 0035 + FF = 0134 sera écrit dans la zone des chaînes et écrasera un octet d'une autre chaîne ***(non corrigé)***.

E9DE/E9EC commande RESUME, lors du rajustement de TXTPTR sur l'instruction ayant causé l'erreur la routine cherche un octet ":" cette procédure est dangereuse car la valeur #3A peut ainsi être le HH d'un n° de ligne et alors bonjour le plantage... ***(non corrigé)***.

EB25/EB90 la commande NUM n'effectue aucune vérification de la validité des paramètres. Il est donc possible de placer dans TRAVNUM (et même dans TRAVPAS) une valeur supérieure à 63999 qui est la limite maximale des n° de ligne BASIC: attention à ce que vous tapez! ***(non corrigé)***.

ED3C/ED51 grosse bogue de LINPUT, qui provoquait un grave problème de gestion du curseur. Ceci apparaît lorsque la longueur de la chaîne demandée dépasse 38 caractères. Les facéties du curseur sont quasi-imprévisibles et rendent impossible l'utilisation, à coup sûr, du paramètre "@x,y,". ***Corrigé*** en ECB5 et EA36.

F210/F233 commande WINDOW: la vérification du type de fichier intervient après son chargement en RAM overlay, en cas d'erreur, SEDORIC a toutes les chances d'être écrasé! ***(non corrigé)***.

F325/F327/F3CA encore la commande WINDOW, cette fois c'est l'existence du tableau WI$ qui n'est pas vérifiée ***(non corrigé)***.

F3F3/F424 gestion de fichiers, routine de vérification de l'existence et création éventuelle de **FI**, cette vérification est plus que cavalière et peut conduire à toutes les catastrophes de plus cette routine, qui est l'une des plus utilisées, se propose à tout moment de créer le pseudo-tableau **FI**, même si elle n'est pas

appelée par une commande OPEN ***(non corrigé)***.

F526/F525 gestion de fichiers, routine de gestion des champs, il est possible d'avoir le même "nom_de_champ(index)" dans plusieurs fichiers, cependant lors d'un transfert de ou vers un champ, grâce aux commandes LSET ou RSET, seul le premier "nom_de_champ(index)" est accessible. Ceci est dû au fait que SEDORIC ne compare pas le NL pour lequel le "nom_de_champ(index)" a été définis avec le NL courant. La vérification du NL et du "nom_de_champ(index)" peut être obtenue en changeant un octet de SEDORIC: remplacer BVC F548 (50 20) par BVC F54B (50 23) en F526 ***(non corrigé)***.

F5FE bogue de la commande SEDORIC “>” ***corrigée*** par RAY.

FA62/FA63 gestion de fichiers, routine d'extension de **FI**, le LDY A1 qui se trouve là est absolument inutile, toutefois, il ne crée aucun problème ***(non corrigé)***.

FC48/FC4A commande FIELD, analyse de syntaxe, il faudrait remplacer le JSR D22E par un JSR D22C, sinon le séparateur de paramètres peut être non seulement une virgule, mais n'importe quoi! ***(non corrigé)***.

Bogues dans le manuel SEDORIC

Le nombre maximum de secteurs par disquette n'est pas de 1200 comme indiqué page 37, mais de 1919.

Bogue page 62 concernant les lignes utilisées par CREATEW. La ligne "service" et la première ligne (n°0) ainsi que la dernière ligne de l'écran (n°26) ne sont pas utilisées.

Commande ERR, en mode direct le n° de ligne est #FFFF (soit 65535 et non 65635 comme indiqué dans le manuel page 59).

Erreur dans le manuel page 60: avec RESUME NEXT, l'exécution est reprise non pas à la ligne suivante, mais à la commande qui suit celle qui a provoqué l'erreur.

Commande USER, le manuel comporte une erreur page 70 dans la syntaxe indiquée: la virgule est indispensable devant DEF (les exemples donnés sont eux corrects), sinon SEDORIC considère alors qu'il s'agit d'un paramètre utilisateur.

Erreur dans le manuel concernant ce que retourne la commande INSTR: renvoie IN = 0 si la chaîne à examiner est vide ou si la chaîne recherchée est vide ou n'est pas trouvée et "ILLEGAL_QUANTITY_ERROR" si la position indiquée est nulle ou supérieure à la longueur de la chaîne à examiner.

LINPUT: l'option ",S" contrairement à ce qui est indiqué dans le manuel page 61, interdit de sortir avec les flèches de déplacement (mode par défaut).

Toujours LINPUT page 61, parmi les caractères de contrôles valides (CTRL/D (double hauteur), CTRL/T (minuscules/MAJUSCULES), CTRL/N (effacement ligne), CTRL/Z (ESC pour attributs vidéo), DEL, ESC (sortie), RETURN (sortie) et flèches (déplacement et sortie), le manuel oublie le CTRL/D.

Dans le préambule concernant la gestion de fichiers, page 76, ainsi que pour les commande OPEN S, OPEN R et OPEN D, pages 77, 82 et 88, le manuel passe sous silence le pseudo-tableau **FI** de type entier

qui est réservé et, beaucoup plus grave, oublie d'indiquer qu'il est interdit de créer un tableau avec la commande DIM dès qu'un OPEN a été utilisé et ceci tant que tous les fichiers ouverts ne sont pas fermés avec CLOSE.

Commande &(), pour les fichiers de type "**S**", cette commande retourne -1 (vrai) dans tous les cas (±n°) si la fin du fichier est atteinte et si ce n'est pas le cas, retourne soit 0 (false) si &(+n°) soit le type d'enregistrement si &(-n°). C'est le contraire de ce qui est indiqué dans le manuel, page 81.

Utilisation de la zone BFE0 à BFFF

Attention, les commandes LINE et BOX utilisent la zone BFE0 à BFFF en RAM. Ceci est un choix malheureux, quasiment assimilable à une bogue, car de nombreux programmes utilisent cette zone pour loger une petite routine en langage machine. Toute utilisation des commandes LINE et BOX entraînera donc l'écrasement de la routine. Il y a gros a parier que l'utilisateur ne comprendra pas ce qui lui arrive! ***(non corrigé)***.

ANNEXE n° 18
Mots clés SEDORIC

page

ACCENT ... 324
APPEND ... 447
AZERTY ... 326
BACKUP ... 51 et 359
BOX ... 350 et 354
BUILD ... 452
CHANGE ... 74 et 359
CHKSUM ... 146 et 316
CLOSE ... 431
COPY ... 89 et 359
CREATEW ... 258
CRESEC ... 415
DEL ... 283
DELBAK ... 283
DELETE ... 43 et 358
DESTROY ... 283
DIR ... 278
DKEY ... 115 et 357
DNAME ... 103 et 358
DNUM ... 108 et 357
DSYS ... 111 et 357
DTRACK ... 104 et 358
ERR ... 311
ERRGOTO ... 312
ERROR ... 313
ESAVE ... 256
EXT ... 144 et 315
FIELD ... 433
FRSEC ... 414
HCUR ... 328
INIST ... 110 et 358
INIT ... 123 et 360
INSTR ... 329
JUMP ... 447
KEY ... 296
KEYDEF ... 233
KEYIF ... 234
KEYSAVE ... 256
KEYUSE ... 232
LCUR ... 327
LDIR ... 302
LINE ... 350 et 351
LINPUT ... 331
LOAD ... 264
LSET ... 438
LTYPE ... 450
LUSING ... 350
MERGE ... 82 et 358
MOVE ... 46 et 358
NUM ... 321
OLD ... 268
OPEN ... 419
OUT ... 297
PMAP ... 413
PR ... 301
PROT ... 154 et 315
PUT ... 416
QUIT ... 303
QWERTY ... 327
RANDOM ... 300
REN ... 287
RENUM ... 30 et 359
RESET ... 301
RESTORE ... 302
RESUME ... 313
REWIND ... 426
RSET ... 439
SAVE ... 252
SAVEM ... 251
SAVEO ... 253
SAVEU ... 251
SEARCH ... 290
SEEK ... 68 et 359
SMAP ... 413
STATUS ... 151 et 315
STRUN ... 305
SWAP ... 316
SYS ... 113 et 359

SYSTEM .... 155 et 316
TAKE 409
TKEN 307
TRACK .... 106 et 358
TYPE 451
UNPROT .... 155 et 316
UNTKEN 308
USER 318
USING 342
VISUHIRES .... 156 et 315
VUSER .... 117 et 357
WIDTH 297
WINDOW 363
"&()" 406
"<" .... 438 et 439
">" 393
"]" 328

ANNEXE n° 19
Codes de fonctions

Fonctions re-définissables

Les 16 premières fonctions sont définies par la table REDEF (de C880 à C97F)

000	#00	espace (=rien)
001	#01	DOKE#2F5,#
002	#02	DOKE#2F5,#467+CR
003	#03	DOKE#2F9,#
004	#04	DOKE#2F9,#D070+CR
005	#05	DOKE#2FC,#
006	#06	DOKE#2FC,#461+CR
007	#07	PAPER0:INK7+CR
008	#08	CALL#F8D0+CR
009	#09	ê (ASCII n°126 = #7E)
010	#0A	?HEX$(PEEK(#
011	#0B	?HEX$(DEEK(#
012	#0C	PEEK(#
013	#0D	DEEK(#
014	#0E	POKE#
015	#0F	DOKE#

Fonctions pré-définies

Les 16 fonctions suivantes sont définies par la table PREDEF (de C980 à C9DD)

016	#10	HEX$(
017	#11	CALL#
018	#12	TEXT
019	#13	FORI=1TO
020	#14	LEFT$(
021	#15	MID$(
022	#16	RIGHT$(
023	#17	STR$(
024	#18	UNPROT
025	#19	© (ASCII n° 96 = #60)
026	#1A	USING
027	#1B	VISUHIRES"
028	#1C	VUSER
029	#1D	WIDTH
030	#1E	WINDOW
031	#1F	!RESTORE

Mots clés du DOS

Ce sont les commandes de SEDORIC.

032	#20	APPEND
033	#21	APPEND
034	#22	AZERTY
035	#23	ACCENT
036	#24	BOX
037	#25	BACKUP
038	#26	BUILD
039	#27	CHANGE
040	#28	CLOSE
041	#29	COPY
042	#2A	CREATEW
043	#2B	CRESEC
044	#2C	CHKSUM
045	#2D	DELETE
046	#2E	DESTROY
047	#2F	DELBAK
048	#30	DEL
049	#31	DIR
050	#32	DTRACK
051	#33	DNUM
052	#34	DNAME
053	#35	DKEY
054	#36	DSYS
055	#37	DTRACK
056	#38	ERRGOTO
057	#39	ERRGOTO
058	#3A	ERROR
059	#3B	ERROR
060	#3C	ERR
061	#3D	ESAVE
062	#3E	EXT
063	#3F	FIELD
064	#40	FRSEC
065	#41	HCUR
066	#42	INIT
067	#43	INSTR
068	#44	INIST
069	#45	JUMP
070	#46	KEYIF
071	#47	KEYIF
072	#48	KEYUSE
073	#49	KEYDEF
074	#4A	KEYDEF
075	#4B	KEYSAVE

076	#4C	KEY
077	#4D	LINE
078	#4E	LSET
079	#4F	LUSING
080	#50	LUSING
081	#51	LINPUT
082	#52	LINPUT
083	#53	LOAD
084	#54	LDIR
085	#55	LTYPE
086	#56	LCUR
087	#57	MOVE
088	#58	MERGE
089	#59	NUM
090	#5A	OUT
091	#5B	OLD
092	#5C	OPEN
093	#5D	PUT
094	#5E	PROT
095	#5F	PR
096	#60	PMAP
097	#61	QUIT
098	#62	QWERTY
099	#63	RESUME
100	#64	RSET
101	#65	REWIND
102	#66	RENUM
103	#67	REN
104	#68	RANDOM
405	#69	RANDOM
406	#6A	RESTORE
107	#6B	RESET
108	#6C	SWAP
109	#6D	SEEK
110	#6E	STRUN
111	#6F	STRUN
112	#70	SYSTEM
113	#71	STATUS
114	#72	SAVEU
115	#73	SAVEM
116	#74	SAVEO
117	#75	SAVE
118	#76	SEARCH
119	#77	SYS
120	#78	SMAP
121	#79	TKEN
122	#7A	TAKE
123	#7B	TYPE

124	#7C	TRACK
125	#7D	USER
126	#7E	UNTKEN
127	#7F	USING
-	-	UNPROT
-	-	VISUHIRES
-	-	VUSER
-	-	WIDTH
-	-	WINDOW
-	-	RESTORE
-	-	]
-	-	255

<u>Token de la ROM</u>

Ce sont les commandes du BASIC

128	#80	END
129	#81	EDIT
130	#82	STORE
131	#83	RECALL
132	#84	TRON
133	#85	TROFF
134	#86	POP
135	#87	PLOT
136	#88	PULL
137	#89	LORES
138	#8A	DOKE
139	#8B	REPEAT
140	#8C	UNTIL
141	#8D	FOR
142	#8E	LLIST
143	#8F	LPRINT
144	#90	NEXT
145	#91	DATA
146	#92	INPUT
147	#93	DIM
148	#94	CLS
149	#95	READ
150	#96	LET
151	#97	GOTO
152	#98	RUN
153	#99	IF
154	#9A	RESTORE
155	#9B	GOSUB
156	#9C	RETURN
157	#9D	REM ('=39)

158	#9E	HIMEM
159	#9F	GRAB
160	#A0	RELEASE
161	#A1	TEXT
162	#A2	HIRES
163	#A3	SHOOT
164	#A4	EXPLODE
165	#A5	ZAP
166	#A6	PING
167	#A7	SOUND
168	#A8	MUSIC
169	#A9	PLAY
170	#AA	CURSET
171	#AB	CURMOV
172	#AC	DRAW
173	#AD	CIRCLE
174	#AE	PATTERN
175	#AF	FILL
176	#B0	CHAR
177	#B1	PAPER
178	#B3	INK
179	#B3	STOP
180	#B4	ON
181	#B5	WAIT
182	#B6	CLOAD
183	#B7	CSAVE
184	#B8	DEF
185	#B9	POKE
186	#BA	PRINT
187	#BB	CONT
188	#BC	LIST
189	#BD	CLEAR
190	#BE	GET
191	#BF	CALL
192	#C0	!
193	#C1	NEW
194	#C2	TAB(
195	#C3	TO
196	#C4	FN
197	#C5	SPC(
198	#C6	@
199	#C7	AUTO
200	#C8	ELSE
201	#C9	THEN
202	#CA	NOT
203	#CB	STEP
204	#CC	+
205	#CD	-

206	#CE	*
207	#CF	/
208	#D0	^
209	#D1	AND
210	#D2	OR
211	#D3	>
212	#D4	=
213	#D5	<
214	#D6	SGN
215	#D7	INT
216	#D8	ABS
217	#D9	USR
218	#DA	FRE
219	#DB	POS
220	#DC	HEX$
221	#DD	&
222	#DE	SQR
223	#DF	RND
224	#E0	LN
225	#E1	EXP
226	#E2	COS
227	#E3	SIN
228	#E4	TAN
229	#E5	ATN
230	#E6	PEEK
231	#E7	DEEK
232	#E8	LOG
233	#E9	LEN
234	#EA	STR$
235	#EB	VAL
236	#EC	ASC
237	#ED	CHR$
238	#EE	PI
239	#EF	TRUE
240	#F0	FALSE
241	#F1	KEY$
242	#F2	SCRN
243	#F3	POINT
244	#F4	LEFT$
245	#F5	RIGHT$
246	#F6	MID$
247	#F7	
248	#F8	
249	#F9	
250	#FA	
251	#FB	
252	#FC	
253	#FD	

Et enfin

254	#FE	DEL
255	#FF	Génération des numéros de lignes

ANNEXE n° 20
Futures Extensions

Place disponible pour de nouvelles implémentations

Entre parenthèse est indiqué le nombre de NOPs occupant l'espace disponible.

Dans le NOYAU:
DA4F (1), DA8D (1), E406-E408 (3), E6E5-E70A (38), EA06-EA2F (42), F608-F609 (2), F638-F63F (8)

Dans la BANQUE5:
C4AD-C4D4 (40), C793-C79F (13) et C7BA-C7FF (70)

Dans la BANQUE6:
C64F (1) et C65D-C65F (3)

Dans la BANQUE7:
C5F6 (1)

Il existe encore quelques zones potentiellement libres, sous réserve de vérifier qu'elles ne servent effectivement à rien:

Dans la BANQUE0:
C599-C5FF (105 octets)

Dans la BANQUE2:
C7F6-C7FF (10 octets)

Dans la BANQUE4:
C7F8-C7FF (8 octets)

ET SI CELA NE SUFFIT PAS...

Vous pouvez sans problème déplacer dans une nouvelle BANQUE les deux commandes STRUN et TKEN (bloc de E853 à E8D5, soit 131 octets). Pour ces deux commandes, le mode direct n'est pas autorisé, ce qui veut dire qu'elles ne sont utilisables qu'en mode programme. Il faut laisser en place la dernière routine (XMAJSTR copie l'adresse et la longueur d'une chaîne alphanumérique (les 3 octets D0, D1 et D2) en B6, B7 et B8) située de E8D6 à E8E0, qui est utilisée par d'autres commandes (UNTKEN et USING). Le déplacement de ces deux commandes ne pose pas de problème, à condition de suivre les indications ci-dessous.

Avec un SEDORIC patché (voir L'ANNEXE concernant le PATCH.001), l'appel à ces commandes se fera de manière transparente pour l'utilisateur, au prix d'une légère pause dans l'exécution du programme.

Pour ajouter une nouvelle commande

1) Formatez une disquette SEDORIC V3.0 Master: INIT A,17,42,S vous devez obtenir 607 free sectors et 0 files. Ce sera votre disquette cible, elle s'appellera SEDO3A. Préparez une seconde disquette avec vos outils préférés (moniteur, éditeur de disquette etc..) Ce sera votre disquette de travail, elle s'appellera SEDO3B. Effectuez une copie de SEDO3A que vous appellerez SOS. Placez SEDO3A dans le drive A et SEDO3B dans le drive B et re-bootez (sinon, adaptez les indications à votre configuration).

2) Récupérez les fichiers systèmes existants:

SAVE"A-NOYAU",A#1400,E#17FF	ce qui fera	61 secteurs
SAVE"BANQUE1",A#C400,E#C7FF		5 secteurs
SAVE"BANQUE2",A#C400,E#C7FF		5 secteurs
SAVE"BANQUE3",A#C400,E#C7FF		5 secteurs
SAVE"BANQUE4",A#C400,E#C7FF		5 secteurs
SAVE"BANQUE5",A#C400,E#C7FF		5 secteurs
SAVE"BANQUE6",A#C400,E#C7FF		5 secteurs
SAVE"BANQUE7",A#C400,E#C7FF		5 secteurs

Votre directory indique maintenant 511 secteurs libres et 8 fichiers. Maintenant, il suffit de modifier les coordonnées des descripteurs de ces fichiers pour qu'ils correspondent aux fichiers système. Pour cela, à l'aide d'un éditeur de disquette, allez dans le secteur 4 de la piste 20 (#14). Vous y trouverez 8 lignes de 16 octets (de n° #00 à #0F) correspondant aux 8 fichiers sauvés. Pour chaque ligne, modifiez les octets n° #0C et #0D comme suit:

Pour NOYAU:	remplacez	05 0A	par	00 04
Pour BANQUE1:		09 03	03 0E	
Pour BANQUE2:		09 08	04 02	
Pour BANQUE3:		09 0D	04 07	
Pour BANQUE4:		0A 01	04 0C	
Pour BANQUE5:		0A 06	04 11	
Pour BANQUE6:		0A 0B		05 05
Pour BANQUE7:		0A 10	05 0A	

Sauvez et à l'aide de votre éditeur de disquette, copiez la première page de bitmap (secteur 2 de la piste 20) de la disquette SOS sur la disquette SEDO3A. Idem pour la deuxième page (secteur 3 de la piste 20). Le directory de SEDO3A doit maintenant indiquer 607 secteurs libres (comme au départ) et 8 fichiers (ils étaient présents mais invisibles au directory).

3) Elaborez votre nouvelle BANQUE: SAVE"BANQUE8",A#C400,E#C7FF soit 5 secteurs. Le directory doit maintenant indiquer 602 secteurs libres et 9 fichiers. Comme précédemment modifiez les coordonnées du descripteur: remplacez 0B 04 par 05 0F A l'aide de votre moniteur favori, placer le code correspondant à votre (vos) nouvelle(s) commande(s) dans ce fichier en respectant les 2 points suivants:

-Les 4 premiers octets (C400 à C403) sont réservés pour vectoriser les messages

-Toutes les commandes présentes dans la BANQUE doivent avoir leur entrée dans la première page de la BANQUE (de C404 à C4FF). Utilisez si besoin des JMP (vectorisation) pour atteindre une entrée placée dans le reste de la BANQUE (de C500 à C7FF). Notez pour chaque commande le LL de

l'adresse de l'entrée dans la page #C400.

4) Greffez vos nouvelles commandes dans la zone EA06 du NOYAU en indiquant le LL de l'adresse de l'entrée dans la page #C400 de chaque commande avec un BIT LL A0 (cachant un LDY#LL) et notez l'adresse de ce LDY pour chaque commande: c'est le point d'entrée de la commande dans le NOYAU. A la suite de tous les BIT LL A0, placez un LDX#65 (le début de la BANQUE n°8 se trouve au #56 ème secteur de la disquette master). Finalement, ajoutez un JMPF15E (routine de gestion des BANQUES).

5) Modifiez de la table des mots-clés: vos nouvelles commandes doivent prendre la place des commandes SEDORIC obsolètes (version en minuscules contenant un mot-clé BASIC) encore libres:

de C9E2 à C9E7	(A)PPEND
de CA5C à CA62	(E)RRGOTO
de CA63 à CA67	(E)RROR
de CA9B à CA9F	(K)EYIF
de CAA6 à CAAB	(K)EYDEF
de CAC2 à CAC7	(L)USING
de CACE à CAD3	(L)INPUT
de CB2E à CB33	(R)ANDOM
de CB34 à CB3A	(R)ESTORE
de CB48 à CB4C	(S)TRUN

Respecter l'initiale et la longueur de chaque mot-clé. Il est possible d'utiliser une initiale contiguë comme cela a été fait par exemple avec USING pour VISUHIRES avec éventuellement un déplacement de certaines commandes. Pour toutes les modifications, notez l'adresse de début du groupe de mots-clés de même initiale, le n° d'ordre du premier mot-clé et le nombre de mots clés dans ce groupe.

6) Modifiez la table des initiales: Vérifiez que pour chacune des initiales altérées, les 4 octets qui la caractérisent sont corrects, sinon corrigez (adresse de début du groupe de mots-clés de même initiale dans la table des mots-clés, n° d'ordre du premier mot-clé et le nombre de mots clés dans ce groupe).

7) Modifiez la table des adresses d'exécution: pour chaque commande modifiée ou déplacée, vérifiez que l'adresse d'exécution est correcte, sinon corrigez en indiquant l'adresse que vous avez notée au §4.

Restriction dans l'utilisation de la plage BFE0 à BFFF en RAM

Attention, les commandes LINE et BOX utilisent la zone BFE0 à BFFF en RAM. Ceci est un choix malheureux, quasiment assimilable à une bogue, car de nombreux programmes utilisent cette zone pour loger une petite routine en langage machine. Toute utilisation des commandes LINE et BOX entraînera donc l'écrasement de la routine. Il y a gros a parier que l'utilisateur ne comprendra pas ce qui lui arrive!

ANNEXE n° 21

Routines d'intérêt général
(par ordre chronologique)

CFCD	XRWTS accès à la routine de gestion des lecteurs. X contient la commande. En sortie, Z = 1 si pas d'erreur, Z = 0 sinon. V = 1 si la disquette est protégée en écriture. DRIVE, PISTE, SECTEUR, et RWBUF doivent être à jour.
D0A5	Handler d' IRQ (sous-programme vectorisé en FFFE)
D0EA	Lit le numéro de piste sous la tête.
D121	NMI sous-programme vectorisé en FFFA.
D136	affiche "LFCRBREAK_ON_BYTE_#".
D154	JSR C4A0/ROM retourne au Ready après affichage d'un message d'erreur.
D15C	JSR C3F4/ROM décale un bloc mémoire vers le haut. En CE/CF adresse du premier octet du bas, en C9/CA adresse dernier octet du haut, en C7/C8 et AY adresse cible vers haut, "OUT_OF_MEMORY_ERROR" si adresse cible > adresse du bas des chaînes A2/A3 revient avec nouveau début - #100 en C7/C8 et nouvelle fin en A0/A1 (haut des tableaux).
D164	JSR C444/ROM vérifie que l'adresse AY est en dessous des chaînes. "OUT_OF_MEMORY_ERROR" si AY trop haut, zone C7/CF n'est pas affectée, AY conservé.
D16C	Affiche "OUT_OF_MEMORY_ERROR", puis réinitialise la pile et retourne au "Ready" (JSR C47E puis C496/ROM).
D16F	Affiche "DISP_TYPE_MISMATCH_ERROR", puis réinitialise la pile et retourne au "Ready" (JSR C47E puis C496/ROM).
D178	JSR C496/ROM affiche "_ERROR", puis réinitialise la pile et retourne au "Ready".
D180	JSR C4A8/ROM retourne au "Ready".
D188	JSR C563/ROM restaure les liens des lignes à partir du début.
D18C	JSR C563/ROM restaure les liens des lignes à partir de l'adresse AY.
D194	JSR C5FA/ROM encode les mots-clés.

D19C JSR C6B3/ROM recherche une ligne BASIC selon le n° en 33/34 à partir du début. Si trouve, retourne avec C = 1 et adresse en CE/CF (visant le premier octet de lien).

D1A4 JSR C6C3/ROM recherche une ligne BASIC à partir de la ligne courante. Si trouve, retourne avec C = 1 et adresse en CE/CF (visant le premier octet de lien).

D1AC JSR C73A/ROM place TXTPTR au début du programme BASIC.

D1B4 JSR C76C/ROM exécute la commande "LIST" simplifiée.

D1BC JSR C816/ROM met l'imprimante en service et inhibe l’affichage sur l’écran. Cette routine ne marche qu’avec la ROM 1.1: un simple RTS est exécuté avec la ROM 1.0.

D1C4 JSR C82F/ROM met l'imprimante hors service et restaure l’affichage sur l’écran.

D1CC JSR C952/ROM exécute la commande "RESTORE" du BASIC.

D1D4 JSR CA23/ROM génère un "UNDEF'D_STATEMENT_ERROR" (GOSUB).

D1DC JSR CA4E & CA3F/ROM calcule le déplacement à l'instruction suivante, met à jour TXTPTR en ajoutant Y.

D1EB JSR CA73/ROM exécute la commande "IF".

D1F3 JSR D39E/RAM overlay puis JSR CAE2/ROM relit le caractère à TXTPTR, le convertit en MAJUSCULE puis évalue le numéro de ligne à TXTPTR (résultat en 33/34).

D1FE JSR CB39/ROM affecte un nombre à une variable.

D206 JSR CBF0/ROM va à la ligne.

D20E JSR CCD9/ROM affiche le caractère présent dans A.

D216 JSR CF17/ROM évalue une expression numérique à TXTPTR. Retourne avec la valeur numérique dans ACC1.

D219 JSR CF09/ROM vérifie que l'expression évaluée à TXTPTR est bien numérique.

D21B JSR CF09/ROM vérifie que l'expression évaluée à TXTPTR est bien alphanumérique.

D21C JSR CF09/ROM vérifie que l'expression évaluée à TXTPTR est bien conforme.

D224 JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne.

D22C JSR D067/ROM puis D3A1/RAM overlay exige une virgule à TXTPTR et lit le caractère suivant avec conversion éventuelle de minuscule en MAJUSCULE. Cette lecture ne sert

souvent qu'à placer TXTPTR sur le caractère qui suit la virgule.

D22E JSR D067/ROM puis D3A1/RAM overlay demande à TXTPTR un octet identique à A et lit le caractère suivant avec conversion éventuelle de minuscule en MAJUSCULE.

D238 JSR D188/ROM décode le nom de la variable à TXTPTR et place "l'adresse" de cette variable dans AY, B6/B7 et D3/D4, cette adresse pointe en fait sur les data (longueur et adresse dans le cas d'une chaîne) de la variable dans la zone des variables BASIC.

D244 JSR D1E8/ROM cherche l'adresse de la valeur d'une variable dont les 2 caractères significatifs sont indiqués en B4/B5.

D24C JSR D2A9/ROM transfère le nombre de ACC1 en D4-D3 (non signé)

D254 JSR D499/ROM transfère le nombre de AY dans ACC1 (signé).

D25C JSR D4D2/ROM interdit le mode direct.

D264 JSR D5AB/ROM réserve une place en mémoire pour une chaîne de longueur A Sauvegarde la longueur en D0 et l'adresse en D1/D2.

D26C JSR D782/ROM génère une "STRING_TOO_LONG_ERROR".

D274 JSR D7D0 et CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien alphanumérique, retourne l'adresse de la chaîne dans XY et 91/92 ainsi que sa longueur dans A.

D27F CF17/ROM, CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X.

D282 JSR D8CB/ROM prend un entier dans ACC1 et le retourne dans X.

D28A JSR D926/ROM convertit le nombre présent dans ACC1 en entier signé dans YA, D3/D4 et 33/34.

D292 JSR DA22/ROM prend 2 coordonnées xy à TXTPTR et les retourne dans 2F8(x) et X(y).

D29A JSR DB0B/ROM effectue AY - ACC1 -> ACC1 (soustraction).

D2A2 JSR DB22/ROM additionne le contenu de ACC1 et la valeur pointée par AY et replace le résultat dans ACC1.

D2AA JSR DCED/ROM multiplie le contenu de ACC1 par la valeur pointée par AY et replace le résultat dans ACC1.

D2B2 JSR DDE4/ROM effectue AY / ACC1 -> ACC1 (division).

D2BA JSR DE7B/ROM transfère dans ACC1 la valeur pointée par AY.

D2C2 JSR DEAD/ROM recopie les 5 octets de ACC1 vers les adresses XY à XY + 4.

D2CA JSR DF40/ROM transfère un nombre non signé YA dans ACC1.

D2D2 JSR E0D5/ROM convertit ACC1 en chaîne décimale d'adresse AY.

D2DA JSR E271/ROM effectue un changement de signe de ACC1.

D2E2 JSR E37D/ROM génère un nombre entre 0 et 1 (en FA).

D2EA JSR E38B/ROM effectue la fonction ACC1 = COS(ACC1).

D2F2 JSR E392/ROM effectue la fonction ACC1 = SIN(ACC1).

D2FA JSR E853/ROM évalue un nombre non signé à TXTPTR (sur 2 octets).

D302 JSR EB78/ROM saisit une touche: si touche frappée alors N = 1 et A = code ASCII sinon N = 0.

D30A JSR EDE0/ROM autorise IRQ (gestion clavier et curseur).

D312 JSR F110/ROM exécute la commande "DRAW".

D31A JSR F4EF/ROM trouve le code ASCII de la touche pressée. En entrée, 0208 contient le code de la touche, 0209 le code de la touche SHIFT ou CTRL et 020C le masque minuscule/MAJUSCULE. En sortie A contient le code ASCII avec b7 à 1. Si le b7 de A est à 0, pas de touche pressée.

D322 JSR F590/ROM appelle la routine d' E/S du PSG 8912. Met X dans le registre A du PSG 8912 (Programmable Sound Generator).

D32A JSR F801/ROM éteint/allume le curseur. Si le curseur était visible (b0 de 026A à 1) et si A = #01 le curseur sera mis en vidéo inverse sinon le caractère sous le curseur sera en vidéo normale.

D332 JSR F982/ROM régénère le jeu de caractères normaux (descend de la ROM dans la RAM).

D33A JSR 00E2/ROM incrémente TXTPTR et lit un caractère (CHRGET). Les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés.

D342 JSR 00E8/ROM lit le caractère à TXTPTR (CHRGOT). Les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés.

D34A Copie NOM et EXT de la table CCF7 dans BUFNOM..

D35C Affiche (X+1) ème message d'erreur externe terminé par un "caractère + 128".

D364 XAFSC affiche le (X+1) ème message externe terminé par "caractère + 128", EXTMS doit contenir l'adresse - 1 du premier message.

D36C Affiche le (X+1) ème message situé en CEE7 et terminé par un "caractère + 128".

D372 Affiche le (X+1) ème message situé en CDBF et terminé par un "caractère + 128".

D376 Entrée réelle affichage (X+1) ème message de zone AY+1 terminé par un “caractère + 128".

D398 XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés (identique au CHRGET du BASIC, mais en plus convertit les minuscules en MAJUSCULES).

D39E XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés (identique au CHRGOT du BASIC, mais en plus convertit les minuscules en MAJUSCULES).

D3A1 XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A.

D44F XNF lit un nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM.

D451 XNFA lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM.

D5D8 XROM exécute à partir de la RAM une routine ROM. Le JSR XROM doit être suivi dans l'ordre de l'adresse de la routine pour la V1.0, puis de l'adresse pour la V1.1.

D60E Convertit n° lecteur en lettre et l'affiche.

D613 XAFHEX affiche en hexadécimal le contenu de A.

D62A XAFCAR affiche le caractère ASCII contenu dans A.

D637 XAFSTR affiche une chaîne terminée par 0 et dont l'adresse est donnée par AY.

D648 Affiche "_DISC_IN_DRIVE_ "lettre du lecteur actif"AND_PRESS_’RETURN’" puis demande un "ESC" (C = 1) ou un "RETURN" (C = 0).

D669 Demande un "ESC" (C = 1) ou un "RETURN" (C = 0).

D676 Idem mais élimine l'adresse de retour si "ESC".

D67E Initialise n° erreur et continue à ERRVEC (incrémente X et traite erreur n° X).

D685 Routine de traitement des erreurs.

D6C9 Affiche l'erreur, ré-initialise la pile et retourne au "Ready".

D73E XCURON rend le curseur visible (= vidéo inverse).

D740 XCUROFF cache le curseur (= vidéo normale).

D74E Affiche en décimal sur 2 digits un nombre A de #00 à #63 (0 à 99).

D753 Affiche en décimal sur 5 digits un nombre AY de #0000 à #FFFF (0 à 65535).

D756 Affiche en décimal sur 4 digits un nombre AY de #0000 à #270F (0 à 9999).

D758 Affichage en décimal sur X + 2 digits d'un nombre AY (entrée générale).

D79E XNOWILD recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou génère une "WILDCARD(S)_NOT_ALLOWED_ERROR" si trouvé.

D7BD Vérifie si drive demandé est "on line" et le valide "actif", si non génère une erreur.

D7C0 Vérifie si le drive Y est "on line", si oui le valide "actif", si non génère une erreur.

D7C9 Recherche et met à jour les variables système.

D843 XLKEY prend un caractère au clavier (entrée spéciale LINPUT).

D845 XKEY prend un caractère au clavier (entrée générale).

DA4C XPMAP prend le premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag”première bitmap chargée”).

DA50 Charge le secteur de bitmap de coordonnées AY dans BUF2 et vérifie le format.

DA5D XPBUF1 charge dans BUF1 le secteur Y de la piste A.

DA60 XPBUF2 charge dans BUF2 le secteur Y de la piste A.

DA63 XPBUF3 charge dans BUF3 le secteur Y de la piste A.

DA65 Charge à la page X le secteur Y de la piste A.

DA6D XPAY charge dans RWBUF le secteur Y de la piste A.

DA73 XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF.

DA82 XSCAT sauve le secteur de catalogue contenu dans BUF3, selon POSNMP et POSNMS.

DA8A Ancienne routine XSMAP (sauve le secteur de bitmap sur la disquette), a été déportée en DC80.

DA91 XSBUF1 sauve BUF1 au secteur Y de la piste A.

DA94 XSBUF3 sauve BUF3 au secteur Y de la piste A.

DA96 Sauve la page X dans le secteur Y de la piste A.

DA9E XSAY sauve la page indiquée par RWBUF dans le secteur Y de la piste A.

DAA4 XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF.

DAA8 Sauve BUF1 selon DRIVE, PISTE et SECTEUR.

DAB4 Affiche le nom de fichier présent à POSNMX dans BUF3.

DAC3 Lit Y caractères à POSNMX dans BUF3 et les affiche.

DACE XVBUF1 remplit BUF1 de zéros.

DAD1 XVBUF2 rempli BUF2 de zéros.

DAD4 XVBUF3 rempli BUF3 de zéros.

DAD6 Rempli de zéros une page mémoire à partir de HH = A et LL=#00.

DAE5 Charge dans BUF3 le secteur pointé par POSNMP et POSNMS puis exécute XBUCA.

DAEE XBUCA transfère le nom de fichier contenu dans BUFNOM dans le secteur de catalogue contenu dans BUF3, à la position POSNMX (pour mise à jour de "l'entrée" de catalogue sur la disquette).

DAFE Charge dans BUF3 le secteur pointé par POSNMP et POSNMS puis exécute XCABU.

DB07 XCABU transfère dans BUFNOM le nom de fichier contenu dans le secteur de catalogue placé dans BUF3, à la position POSNMX..

DB17 Comparaison du nom cherché (BUFNOM) et du nom pointé par X dans le catalogue (BUF3).

DB2D Vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé.

DB30 XTVNM cherche sur le lecteur courant le fichier dont le nom est indiqué dans BUFNOM. A la sortie, POSNMX, POSNMP, et POSNMS contiennent la position du nom dans le catalogue (BUF3), et Z = 1 si le fichier n'est pas trouvé.

DB41 Ajuste POSNMX sur “l’entrée” suivante du catalogue et reprend la recherche dans le catalogue du fichier indiqué dans BUFNOM (Z = 1 si fini).

DB59 XTRVCA cherche une place libre dans le catalogue. A la sortie, POSNMX, POSNMP et POSNMS indiquent la position de la place réservée.

DBA5 Cherche le POSNMX de la première place libre dans le directory.

DBC0 XWDESC écrit le ou les descripteurs du fichier à sauver. Revient avec le nombre de secteurs à sauver dans NSSAV (C05A/5B), les coordonnées du premier secteur descripteur dans PSDESC (C05C/5D), le nombre de descripteurs utilisés dans NSDESC (C05E) et premier descripteur en place.

DC6C XLIBSE cherche un secteur libre dans la bitmap dans BUF2, retourne avec A = n° de piste et Y = n° de secteur (sinon "DISK_FULL_ERROR").

DC7D Ancienne routine “Cherche un secteur libre”, déportée en E67F, pour tenir compte de la double bitmap.

DC80 XSMAP sauve la bitmap sur la disquette.

DC89 Ecrit BUF2 dans le premier secteur de bitmap sur la disquette.

DC8B Ecrit BUF2 dans le second secteur de bitmap sur la disquette (entrée secondaire avec Y = #03 à pré-positionner).

DD15 XDETSE libère le secteur Y de la piste A sur la bitmap courante dans BUF2 et incrémente le nombre de secteurs libres. Retourne avec C = 1 si ce secteur était déjà libre. Ne pas oublier de sauver le plus tôt possible cette nouvelle bitmap avec SMAP.

DD2D XCREAY crée une table piste secteur de AY secteurs, en fait marque dans la bitmap en BUF2 que le secteur AY est occupé. En sortie, C = 1 si ce secteur était déjà occupé sinon avec C = 0.

DE28 XDEFSA positionne les valeurs par défaut pour XSAVEB (en fait, positionne pour sauver le programme BASIC).

DE9C XSAVEB sauve le fichier de nom contenu dans BUFNOM, selon VSALO0, VSALO1, DESALO, FISALO, EXSALO.

DFDE XVERTXT vérifie que l’on est bien en mode TEXT, sinon génère une "DISP_TYPE_MISMATCH_ ERROR", ré-initialise la pile et retourne au "Ready".

DFE6 XDEFLO positionne les valeurs par défaut pour XLOADA.

E0E5 XLOADA charge le fichier dont le nom est dans BUFNOM, selon VSALO0, VSALO1, DESALO.

E0EA Charge un fichier selon X = POSNMX, POSNMP et POSNMS, VSALO0, VSALO1, DESALO.

E266 XNOMDE détruit le fichier indexé par POSNMX, dont le secteur de catalogue est dans BUF3 (en fait, tout est positionné comme après un XTVCAT).

E322 Affiche nom_de_fichier et taille du fichier à POSNMX.

E60D Valide le drive s'il est indiqué à TXTPTR, sinon valide DRVDEF.

E635 Ecrit BUF2 dans le second secteur de bitmap sur la disquette.

E63A Entrée de la nouvelle routine XSMAP qui écrit BUF2 dans la deuxième page de bitmap sur la disquette et charge ensuite la première page dans BUF2.

E63C Entrée de la nouvelle routine XSMAP qui écrit BUF2 dans la première page de bitmap sur la disquette et charge ensuite la deuxième page dans BUF2.

E8D6 XMAJSTR copie l'adresse et la longueur d'une chaîne alphanumérique (les 3 octets D0, D1 et D2) en B6, B7 et B8.

E94D XSETOFF Teste si le paramètre à TXTPTR est SET ou OFF, retourne avec C = 1 si SET ou C = 0 si OFF sinon génère une "SYNTAX_ERROR".

EBA3 XCHAR sélectionne le jeu de caractères correct ("normal" ou "accentué"), selon MODCLA.

EC17 XSTATUS initialise PAPER, INK, mode clavier et status console.

ED36 XLINPU routine principale de LINPUT (routine de saisie de chaîne), au retour F4 contient le mode de sortie et D0, D1, D2 donne la longueur et l'adresse de la chaîne dans la zone de stockage des chaînes sous HIMEM..

EE69 XAFXGAU affiche X fois "flèche gauche".

EE73 XAF1GAU affiche une "flèche gauche".

EE76 XAF1DR affiche une "flèche droite".

EE8E XCSTR copie la longueur et l'adresse d'une chaîne alphanumérique (les 3 octets D0, D1 et D2) "dans" la variable BASIC pointée en B8, B9 et BA.

F070 XVERHRS vérifie si on est bien en mode HIRES, sinon génère une "DISP_TYPE_MISMATCH_ ERROR", ré-initialise la pile et retourne au "Ready".

F1E5 XDLOAD charge X secteurs pris à partir du secteur Y de la piste indiquée en C001 (PISTE) et les copie en RAM à partir de la page A.

F3F3 Vérifie l'existence du "pseudo-tableau" FI au début des tableaux et le crée s'il n'existe pas encore.

F4A8 Place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début

du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "S/R/D") puis retourne avec Y = #00.

FD46 Sauve sur la disquette le secteur du fichier qui est présent dans le "General Buffer".

FDD9 Lit l'enregistrement suivant du fichier: Si la fin du fichier n'est pas atteinte, copie un enregistrement complet (type, longueur et valeur de la variable) du "Channel's own Data Buffer" (charge un second secteur dans le buffer si nécessaire), vers le "General Buffer", 06/07 pointe sur le type, la longueur et la valeur du data.

FE38 Ecrit l'enregistrement du "General Buffer" dans le "Channel's own Data Buffer" en utilisant le secteur suivant si nécessaire.

FF3D XGETCAR attend un caractère au clavier et revient avec ce caractère dans A.

ANNEXE n° 22

Routines d’intérêt général

(par thèmes)

AFFICHAGE et IMPRESSION

D136 Affiche "LFCRBREAK_ON_BYTE_#".

D154 JSR C4A0/ROM retourne au Ready après affichage d'un message d'erreur.

D16C Affiche "OUT_OF_MEMORY_ERROR", puis réinitialise la pile et retourne au "Ready" (JSR C47E puis C496/ROM).

D16F Affiche "DISP_TYPE_MISMATCH_ERROR", puis réinitialise la pile et retourne au "Ready" (JSR C47E puis C496/ROM).

D178 JSR C496/ROM affiche "_ERROR", puis réinitialise la pile et retourne au "Ready".

D180 JSR C4A8/ROM retourne au "Ready".

D1BC JSR C816/ROM met l'imprimante en service et inhibe l’affichage sur l’écran. Cette routine ne marche qu’avec la ROM 1.1: un simple RTS est exécuté avec la ROM 1.0.

D1C4 JSR C82F/ROM met l'imprimante hors service et restaure l’affichage sur l’écran.

D1D4 JSR CA23/ROM génère un "UNDEF'D_STATEMENT_ERROR" (GOSUB).

D206 JSR CBF0/ROM va à la ligne.

D20E JSR CCD9/ROM affiche le caractère présent dans A.

D26C JSR D782/ROM génère une "STRING_TOO_LONG_ERROR".

D32A JSR F801/ROM éteint/allume le curseur. Si le curseur était visible (b0 de 026A à 1) et si A = #01 le curseur sera mis en vidéo inverse sinon le caractère sous le curseur sera en vidéo normale.

D332 JSR F982/ROM régénère le jeu de caractères normaux (descend de la ROM dans la RAM).

D35C Affiche (X+1) ème message d'erreur externe terminé par un "caractère + 128".

D364 XAFSC affiche le (X+1) ème message externe terminé par "caractère + 128", EXTMS doit contenir l'adresse - 1 du premier message.

D36C Affiche le (X+1) ème message situé en CEE7 et terminé par un "caractère + 128".

D372 Affiche le (X+1) ème message situé en CDBF et terminé par un "caractère + 128".

D3A1 XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A.

D376 Entrée réelle affichage (X+1) ème message de zone AY+1 terminé par un “caractère + 128".

D60E Convertit n° lecteur en lettre et l'affiche.

D613 XAFHEX affiche en hexadécimal le contenu de A.

D62A XAFCAR affiche le caractère ASCII contenu dans A.

D637 XAFSTR affiche une chaîne terminée par 0 et dont l'adresse est donnée par AY.

D648 Affiche "_DISC_IN_DRIVE_ "lettre du lecteur actif"AND_PRESS_’RETURN’" puis demande un "ESC" (C = 1) ou un "RETURN" (C = 0).

D6C9 Affiche l'erreur, réinitialise la pile et retourne au "Ready".

D73E XCURON rend le curseur visible (= vidéo inverse).

D740 XCUROFF cache le curseur (= vidéo normale).

D74E Affiche en décimal sur 2 digits un nombre A de #00 à #63 (0 à 99).

D753 Affiche en décimal sur 5 digits un nombre AY de #0000 à #FFFF (0 à 65535).

D756 Affiche en décimal sur 4 digits un nombre AY de #0000 à #270F (0 à 9999).

D758 Affichage en décimal sur X + 2 digits d'un nombre AY (entrée générale).

DAB4 Affiche le nom de fichier présent à POSNMX dans BUF3.

E322 Affiche nom_de_fichier et taille du fichier à POSNMX.

E94D XSETOFF Teste si le paramètre à TXTPTR est SET ou OFF, retourne avec C = 1 si SET ou C = 0 si OFF sinon génère une "SYNTAX_ERROR".

EBA3 XCHAR sélectionne le jeu de caractères correct (“normal” ou “accentué”), selon MODCLA.

EE69 XAFXGAU affiche X fois "flèche gauche".

EE73 XAF1GAU affiche une "flèche gauche".

EE76 XAF1DR affiche une "flèche droite".

CONSOLE (CLAVIER et ECRAN)

D1BC JSR C816/ROM met l'imprimante en service et inhibe l'affichage sur l'écran. Cette routine ne marche qu'avec la ROM 1.1: un simple RTS est exécuté avec la ROM 1.0.

D1C4 JSR C82F/ROM met l'imprimante hors service et restaure l'affichage sur l'écran.

D25C JSR D4D2/ROM interdit le mode direct.

D292 JSR DA22/ROM prend 2 coordonnées xy à TXTPTR et les retourne dans 2F8(x) et X(y).

D302 JSR EB78/ROM saisit une touche: si touche frappée alors N = 1 et A = code ASCII sinon N = 0.

D30A JSR EDE0/ROM autorise IRQ (gestion clavier et curseur).

D31A JSR F4EF/ROM trouve le code ASCII de la touche pressée. En entrée, 0208 contient le code de la touche, 0209 le code de la touche SHIFT ou CTRL et 020C le masque minuscule/MAJUSCULE. En sortie A contient le code ASCII avec b7 à 1. Si le b7 de A est à 0, pas de touche pressée.

D322 JSR F590/ROM appelle la routine d' E/S du PSG 8912. Met X dans le registre A du PSG 8912 (Programmable Sound Generator).

D32A JSR F801/ROM éteint/allume le curseur. Si le curseur était visible (b0 de 026A à 1) et si A = #01 le curseur sera mis en vidéo inverse sinon le caractère sous le curseur sera en vidéo normale.

D332 JSR F982/ROM régénère le jeu de caractères normaux (descend de la ROM dans la RAM).

D33A JSR 00E2/ROM incrémente TXTPTR et lit un caractère (CHRGET). Les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés.

D342 JSR 00E8/ROM lit le caractère à TXTPTR (CHRGOT). Les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés.

D398 XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés (identique au CHRGET du BASIC, mais en plus convertit les minuscules en MAJUSCULES).

D39E XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0

si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés (identique au CHRGOT du BASIC, mais en plus convertit les minuscules en MAJUSCULES).

D669 Demande un "ESC" (C = 1) ou un "RETURN" (C = 0).

D676 Idem mais élimine l'adresse de retour si "ESC".

D843 XLKEY prend un caractère au clavier (entrée spéciale LINPUT).

D845 XKEY prend un caractère au clavier (entrée générale).

DFDE XVERTXT vérifie que l'on est bien en mode TEXT, sinon génère une "DISP_TYPE_MISMATCH_ ERROR", ré-initialise la pile et retourne au "Ready".

E94D XSETOFF Teste si le paramètre à TXTPTR est SET ou OFF, retourne avec C = 1 si SET ou C = 0 si OFF sinon génère une "SYNTAX_ERROR".

EBA3 XCHAR sélectionne le jeu de caractères correct ("normal" ou "accentué"), selon MODCLA.

EC17 XSTATUS initialise PAPER, INK, mode clavier et status console.

ED36 XLINPU routine principale de LINPUT (routine de saisie de chaîne), au retour F4 contient le mode de sortie et D0, D1, D2 donne la longueur et l'adresse de la chaîne dans la zone de stockage des chaînes sous HIMEM..

F070 XVERHRS vérifie si on est bien en mode HIRES, sinon génère une "DISP_TYPE_MISMATCH_ ERROR", ré-initialise la pile et retourne au "Ready".

FF3D XGETCAR attend un caractère au clavier et revient avec ce caractère dans A.

GESTION des ERREURS

D136 Affiche "LFCRBREAK_ON_BYTE_#".

D154 JSR C4A0/ROM retourne au Ready après affichage d'un message d'erreur.

D15C JSR C3F4/ROM décale un bloc mémoire vers le haut. En CE/CF adresse du premier octet du bas, en C9/CA adresse dernier octet du haut, en C7/C8 et AY adresse cible vers haut, "OUT_OF_MEMORY_ERROR" si adresse cible > adresse du bas des chaînes A2/A3 revient avec nouveau début - #100 en C7/C8 et nouvelle fin en A0/A1 (haut des tableaux).

D164 JSR C444/ROM vérifie que l'adresse AY est en dessous des chaînes. "OUT_OF_MEMORY_ERROR" si AY trop haut, zone C7/CF n'est pas affectée, AY conservé.

D16C Affiche "OUT_OF_MEMORY_ERROR", puis ré-initialise la pile et retourne au "Ready" (JSR C47E puis C496/ROM).

D16F Affiche "DISP_TYPE_MISMATCH_ERROR", puis ré-initialise la pile et retourne au "Ready" (JSR C47E puis C496/ROM).

D178 JSR C496/ROM affiche "_ERROR", puis ré-initialise la pile et retourne au "Ready".

D180 JSR C4A8/ROM retourne au "Ready".

D1D4 JSR CA23/ROM génère un "UNDEF'D_STATEMENT_ERROR" (GOSUB).

D26C JSR D782/ROM génère une "STRING_TOO_LONG_ERROR".

D67E Initialise n° erreur et continue à ERRVEC (incrémente X et traite erreur n° X).

D685 Routine de traitement des erreurs.

D6C9 Affiche l'erreur, ré-initialise la pile et retourne au "Ready".

D79E XNOWILD recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou génère une "WILDCARD(S)_NOT_ALLOWED_ERROR" si trouvé.

D7BD Vérifie si drive demandé est "on line" et le valide "actif", si non génère une erreur.

D7C0 Vérifie si le drive Y est "on line", si oui le valide "actif", si non génère une erreur.

DC6C XLIBSE cherche un secteur libre dans la bitmap dans BUF2, retourne avec A = n° de piste et Y = n° de secteur (sinon "DISK_FULL_ERROR").

DFDE XVERTXT vérifie que l’on est bien en mode TEXT, sinon génère une "DISP_TYPE_MISMATCH_ ERROR", ré-initialise la pile et retourne au "Ready".

E94D XSETOFF Teste si le paramètre à TXTPTR est SET ou OFF, retourne avec C = 1 si SET ou C = 0 si OFF sinon génère une "SYNTAX_ERROR".

F070 XVERHRS vérifie si on est bien en mode HIRES, sinon génère une "DISP_TYPE_MISMATCH_ ERROR", ré-initialise la pile et retourne au "Ready".

OPERATIONS à TXTPTR

D1AC JSR C73A/ROM place TXTPTR au début du programme BASIC.

D1DC JSR CA4E & CA3F/ROM calcule le déplacement à l'instruction suivante, met à jour TXTPTR en ajoutant Y.

D1F3 JSR D39E/RAM overlay puis JSR CAE2/ROM relit le caractère à TXTPTR, le convertit en MAJUSCULE puis évalue le numéro de ligne à TXTPTR (résultat en 33/34).

D216 JSR CF17/ROM évalue une expression numérique à TXTPTR. Retourne avec la valeur

numérique dans ACC1.

D219 JSR CF09/ROM vérifie que l'expression évaluée à TXTPTR est bien numérique.

D21B JSR CF09/ROM vérifie que l'expression évaluée à TXTPTR est bien alphanumérique.

D21C JSR CF09/ROM vérifie que l'expression évaluée à TXTPTR est bien conforme.

D224 JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne.

D22C JSR D067/ROM puis D3A1/RAM overlay exige une virgule à TXTPTR et lit le caractère suivant avec conversion éventuelle de minuscule en MAJUSCULE. Cette lecture ne sert souvent qu'à placer TXTPTR sur le caractère qui suit la virgule.

D22E JSR D067/ROM puis D3A1/RAM overlay demande à TXTPTR un octet identique à A et lit le caractère suivant avec conversion éventuelle de minuscule en MAJUSCULE.

D238 JSR D188/ROM décode le nom de la variable à TXTPTR et place "l'adresse" de cette variable dans AY, B6/B7 et D3/D4, cette adresse pointe en fait sur les data (longueur/adresse dans le cas d'une chaîne) de la variable dans la zone des variables BASIC.

D274 JSR D7D0 et CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien alphanumérique, retourne l'adresse de la chaîne dans XY et 91/92 ainsi que sa longueur dans A.

D27F CF17/ROM, CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X.

D292 JSR DA22/ROM prend 2 coordonnées xy à TXTPTR et les retourne dans 2F8(x) et X(y).

D2FA JSR E853/ROM évalue un nombre non signé à TXTPTR (sur 2 octets).

D33A JSR 00E2/ROM incrémente TXTPTR et lit un caractère (CHRGET). Les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés.

D342 JSR 00E8/ROM lit le caractère à TXTPTR (CHRGOT). Les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés.

D398 XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés (identique au CHRGET du BASIC, mais en plus convertit les minuscules en MAJUSCULES).

D39E XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0

si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés (identique au CHRGOT du BASIC, mais en plus convertit les minuscules en MAJUSCULES).

D3A1 XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A.

D44F XNF lit un nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM.

D451 XNFA lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM.

E60D Valide le drive s'il est indiqué à TXTPTR, sinon valide DRVDEF.

E94D XSETOFF Teste si le paramètre à TXTPTR est SET ou OFF, retourne avec C = 1 si SET ou C = 0 si OFF sinon génère une "SYNTAX_ERROR".

OPERATIONS sur ACC1 (floating point accumulator)

D216 JSR CF17/ROM évalue une expression numérique à TXTPTR. Retourne avec la valeur numérique dans ACC1.

D224 JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne.

D24C JSR D2A9/ROM transfère le nombre de ACC1 en D4-D3 (non signé)

D254 JSR D499/ROM transfère le nombre de AY dans ACC1 (signé).

D282 JSR D8CB/ROM prend un entier dans ACC1 et le retourne dans X.

D28A JSR D926/ROM convertit le nombre présent dans ACC1 en entier signé dans YA, D3/D4 et 33/34.

D29A JSR DB0B/ROM effectue AY - ACC1 -> ACC1 (soustraction).

D2A2 JSR DB22/ROM additionne le contenu de ACC1 et la valeur pointée par AY et replace le résultat dans ACC1.

D2AA JSR DCED/ROM multiplie le contenu de ACC1 par la valeur pointée par AY et replace le résultat dans ACC1.

D2B2 JSR DDE4/ROM effectue AY / ACC1 -> ACC1 (division).

D2BA JSR DE7B/ROM transfère dans ACC1 la valeur pointée par AY.

D2C2 JSR DEAD/ROM recopie les 5 octets de ACC1 vers les adresses XY à XY + 4.

D2CA JSR DF40/ROM transfère un nombre non signé YA dans ACC1.

D2D2 JSR E0D5/ROM convertit ACC1 en chaîne décimale d'adresse AY.

D2DA JSR E271/ROM effectue un changement de signe de ACC1.

D2E2 JSR E37D/ROM génère un nombre entre 0 et 1 (en FA).

D2EA JSR E38B/ROM effectue la fonction ACC1 = COS(ACC1).

D2F2 JSR E392/ROM effectue la fonction ACC1 = SIN(ACC1).

OPERATIONS sur la RAM

D15C JSR C3F4/ROM décale un bloc mémoire vers le haut. En CE/CF adresse du premier octet du bas, en C9/CA adresse dernier octet du haut, en C7/C8 et AY adresse cible vers haut, "OUT_OF_MEMORY_ERROR" si adresse cible > adresse du bas des chaînes A2/A3 revient avec nouveau début - #100 en C7/C8 et nouvelle fin en A0/A1 (haut des tableaux).

D264 JSR D5AB/ROM réserve une place en mémoire pour une chaîne de longueur A Sauvegarde la longueur en D0 et l'adresse en D1/D2.

DACE XVBUF1 remplit BUF1 de zéros.

DAD1 XVBUF2 rempli BUF2 de zéros.

DAD4 XVBUF3 rempli BUF3 de zéros.

DAD6 Rempli de zéros une page mémoire à partir de HH = A et LL=#00.

F1E5 XDLOAD charge X secteurs pris à partir du secteur Y de la piste indiquée en C001 (PISTE) et les copie en RAM à partir de la page A.

COOMMANDES et ROUTINES BASIC

D188 JSR C563/ROM restaure les liens des lignes à partir du début.

D18C JSR C563/ROM restaure les liens des lignes à partir de l’adresse AY.

D194 JSR C5FA/ROM encode les mots-clés.

D19C JSR C6B3/ROM recherche une ligne BASIC selon le n° en 33/34 à partir du début. Si trouve, retourne avec C = 1 et adresse en CE/CF (visant le premier octet de lien).

D1A4 JSR C6C3/ROM recherche une ligne BASIC à partir de la ligne courante. Si trouve, retourne avec C = 1 et adresse en CE/CF (visant le premier octet de lien).

D1AC JSR C73A/ROM place TXTPTR au début du programme BASIC.

D1B4 JSR C76C/ROM exécute la commande "LIST" simplifiée.

D1CC JSR C952/ROM exécute la commande "RESTORE" du BASIC.

D1EB JSR CA73/ROM exécute la commande "IF".

D1FE JSR CB39/ROM affecte un nombre à une variable.

D244 JSR D1E8/ROM cherche l'adresse de la valeur d'une variable dont les 2 caractères significatifs sont indiqués en B4/B5.

D264 JSR D5AB/ROM réserve une place en mémoire pour une chaîne de longueur A Sauvegarde la longueur en D0 et l'adresse en D1/D2.

D2E2 JSR E37D/ROM génère un nombre entre 0 et 1 (en FA).

D312 JSR F110/ROM exécute la commande "DRAW".

D5D8 XROM exécute à partir de la RAM une routine ROM. Le JSR XROM doit être suivi dans l'ordre de l'adresse de la routine pour la V1.0, puis de l'adresse pour la V1.1.

E8D6 XMAJSTR copie l'adresse et la longueur d'une chaîne alphanumérique (les 3 octets D0, D1 et D2) en B6, B7 et B8.

EE8E XCSTR copie la longueur et l'adresse d'une chaîne alphanumérique (les 3 octets D0, D1 et D2) "dans" la variable BASIC pointée en B8, B9 et BA.

ROUTINES SYSTEME

D0A5 Handler d' IRQ (sous-programme vectorisé en FFFE).

D121 NMI sous-programme vectorisé en FFFA.

D30A JSR EDE0/ROM autorise IRQ (gestion clavier et curseur).

D7C9 Recherche et met à jour les variables système.

E60D Valide le drive s'il est indiqué à TXTPTR, sinon valide DRVDEF.

GESTIONS des LECTEURS de DISQUETTES

CFCD XRWTS accès à la routine de gestion des lecteurs. X contient la commande. En sortie, Z = 1 si pas d'erreur, Z = 0 sinon. V = 1 si la disquette est protégée en écriture. DRIVE, PISTE,

SECTEUR, et RWBUF doivent être à jour.

D0EA Lit le numéro de piste sous la tête.

D7BD Vérifie si drive demandé est "on line" et le valide "actif", si non génère une erreur.

D7C0 Vérifie si le drive Y est "on line", si oui le valide "actif", si non génère une erreur.

DA5D XPBUF1 charge dans BUF1 le secteur Y de la piste A.

DA60 XPBUF2 charge dans BUF2 le secteur Y de la piste A.

DA63 XPBUF3 charge dans BUF3 le secteur Y de la piste A.

DA65 Charge à la page X le secteur Y de la piste A.

DA91 XSBUF1 sauve BUF1 au secteur Y de la piste A.

DA94 XSBUF3 sauve BUF3 au secteur Y de la piste A.

DA96 Sauve la page X dans le secteur Y de la piste A.

DC80 XSMAP sauve la bitmap sur la disquette.

DD2D XCREAY crée une table piste secteur de AY secteurs, en fait marque dans la bitmap en BUF2 que le secteur AY est occupé. En sortie, C = 1 si ce secteur était déjà occupé sinon avec C = 0.

E60D Valide le drive s'il est indiqué à TXTPTR, sinon valide DRVDEF.

F1E5 XDLOAD charge X secteurs pris à partir du secteur Y de la piste indiquée en C001 (PISTE) et les copie en RAM à partir de la page A.

CATALOGUE (BUFNOM & POSNMX)

D34A Copie NOM et EXT de la table CCF7 dans BUFNOM..

D44F XNF lit un nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM.

D451 XNFA lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM.

D79E XNOWILD recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou génère une "WILDCARD(S)_NOT_ALLOWED_ERROR" si trouvé.

DA82 XSCAT sauve le secteur de catalogue contenu dans BUF3, selon POSNMP et POSNMS.

DAB4 Affiche le nom de fichier présent à POSNMX dans BUF3.

DAC3 Lit Y caractères à POSNMX dans BUF3 et les affiche.

DAE5 Charge dans BUF3 le secteur pointé par POSNMP et POSNMS puis exécute XBUCA.

DAEE XBUCA transfère le nom de fichier contenu dans BUFNOM dans le secteur de catalogue contenu dans BUF3, à la position POSNMX (pour mise à jour de "l'entrée" de catalogue sur la disquette).

DAFE Charge dans BUF3 le secteur pointé par POSNMP et POSNMS puis exécute XCABU.

DB07 XCABU transfère dans BUFNOM le nom de fichier contenu dans le secteur de catalogue placé dans BUF3, à la position POSNMX..

DB17 Comparaison du nom cherché (BUFNOM) et du nom pointé par X dans le catalogue (BUF3).

DB2D Vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé.

DB30 XTVNM cherche sur le lecteur courant le fichier dont le nom est indiqué dans BUFNOM. A la sortie, POSNMX, POSNMP, et POSNMS contiennent la position du nom dans le catalogue (BUF3), et Z = 1 si le fichier n'est pas trouvé.

DB41 Ajuste POSNMX sur “l’entrée” suivante du catalogue et reprend la recherche dans le catalogue du fichier indiqué dans BUFNOM (Z = 1 si fini).

DB59 XTRVCA cherche une place libre dans le catalogue. A la sortie, POSNMX, POSNMP et POSNMS indiquent la position de la place réservée.

DBA5 Cherche le POSNMX de la première place libre dans le directory.

DE9C XSAVEB sauve le fichier de nom contenu dans BUFNOM, selon VSALO0, VSALO1, DESALO, FISALO, EXSALO.

E0E5 XLOADA charge le fichier dont le nom est dans BUFNOM, selon VSALO0, VSALO1, DESALO.

E266 XNOMDE détruit le fichier indexé par POSNMX, dont le secteur de catalogue est dans BUF3 (en fait, tout est positionné comme après un XTVCAT).

E322 Affiche nom_de_fichier et taille du fichier à POSNMX.

OPERATIONS sur la BITMAP

DA4C XPMAP prend le premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag”première bitmap chargée”).

DA50 Charge le secteur de bitmap de coordonnées AY dans BUF2 et vérifie le format.

DA8A Ancienne routine XSMAP (sauve le secteur de bitmap sur la disquette), a été déportée en DC80.

DC6C XLIBSE cherche un secteur libre dans la bitmap dans BUF2, retourne avec A = n° de piste et Y = n° de secteur (sinon "DISK_FULL_ERROR").

DC7D Ancienne routine “Cherche un secteur libre”, déportée en E67F, pour tenir compte de la double bitmap.

DC80 XSMAP sauve la bitmap sur la disquette.

DC89 Ecrit BUF2 dans le premier secteur de bitmap sur la disquette.

DC8B Ecrit BUF2 dans le second secteur de bitmap sur la disquette (entrée secondaire avec Y = #03 à pré-positionner).

DD15 XDETSE libère le secteur Y de la piste A sur la bitmap courante dans BUF2 et incrémente le nombre de secteurs libres. Retourne avec C = 1 si ce secteur était déjà libre. Ne pas oublier de sauver le plus tôt possible cette nouvelle bitmap avec SMAP.

DD2D XCREAY crée une table piste secteur de AY secteurs, en fait marque dans la bitmap en BUF2 que le secteur AY est occupé. En sortie, C = 1 si ce secteur était déjà occupé sinon avec C = 0.

E266 XNOMDE détruit le fichier indexé par POSNMX, dont le secteur de catalogue est dans BUF3 (en fait, tout est positionné comme après un XTVCAT).

E635 Ecrit BUF2 dans le second secteur de bitmap sur la disquette.

E63A Entrée de la nouvelle routine XSMAP qui écrit BUF2 dans la deuxième page de bitmap sur la disquette et charge ensuite la première page dans BUF2.

E63C Entrée de la nouvelle routine XSMAP qui écrit BUF2 dans la première page de bitmap sur la disquette et charge ensuite la deuxième page dans BUF2.

DIVERSES OPERATIONS de LECTURE / ECRITURE

DA5D XPBUF1 charge dans BUF1 le secteur Y de la piste A.

DA60 XPBUF2 charge dans BUF2 le secteur Y de la piste A.

DA63 XPBUF3 charge dans BUF3 le secteur Y de la piste A.

DA65 Charge à la page X le secteur Y de la piste A.

DA6D XPAY charge dans RWBUF le secteur Y de la piste A.

DA73 XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF.

DA82 XSCAT sauve le secteur de catalogue contenu dans BUF3, selon POSNMP et POSNMS.

DA91 XSBUF1 sauve BUF1 au secteur Y de la piste A.

DA94 XSBUF3 sauve BUF3 au secteur Y de la piste A.

DA96 Sauve la page X dans le secteur Y de la piste A.

DA9E XSAY sauve la page indiquée par RWBUF dans le secteur Y de la piste A.

DAA4 XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF.

DAA8 Sauve BUF1 selon DRIVE, PISTE et SECTEUR.

DAE5 Charge dans BUF3 le secteur pointé par POSNMP et POSNMS puis exécute XBUCA.

DAFE Charge dans BUF3 le secteur pointé par POSNMP et POSNMS puis exécute XCABU.

DB2D Vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé.

DBC0 XWDESC écrit le ou les descripteurs du fichier à sauver. Revient avec le nombre de secteurs à sauver dans NSSAV (C05A/5B), les coordonnées du premier secteur descripteur dans PSDESC (C05C/5D), le nombre de descripteurs utilisés dans NSDESC (C05E) et premier descripteur en place.

DE28 XDEFSA positionne les valeurs par défaut pour XSAVEB (en fait, positionne pour sauver le programme BASIC).

DE9C XSAVEB sauve le fichier de nom contenu dans BUFNOM, selon VSALO0, VSALO1, DESALO, FISALO, EXSALO.

DFE6 XDEFLO positionne les valeurs par défaut pour XLOADA.

E0E5 XLOADA charge le fichier dont le nom est dans BUFNOM, selon VSALO0, VSALO1, DESALO.

E0EA Charge un fichier selon X = POSNMX, POSNMP et POSNMS, VSALO0, VSALO1, DESALO.

E266 XNOMDE détruit le fichier indexé par POSNMX, dont le secteur de catalogue est dans BUF3 (en fait, tout est positionné comme après un XTVCAT).

GESTION des FICHIERS (Séquentiels, diRects et Disques)

F3F3 Vérifie l'existence du "pseudo-tableau" FI au début des tableaux et le crée s'il n'existe pas encore.

F4A8 Place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "S/R/D") puis retourne avec Y = #00.

FD46 Sauve sur la disquette le secteur du fichier qui est présent dans le "General Buffer".

FDD9 Lit l'enregistrement suivant du fichier: Si la fin du fichier n'est pas atteinte, copie un enregistrement complet (type, longueur et valeur de la variable) du "Channel's own Data Buffer" (charge un second secteur dans le buffer si nécessaire), vers le "General Buffer", 06/07 pointe sur le type, la longueur et la valeur du data.

FE38 Ecrit l'enregistrement du "General Buffer" dans le "Channel's own Data Buffer" en utilisant le secteur suivant si nécessaire.

ANNEXE n° 23

Des DRIVES et des DOS

(enquête historique: éléments préliminaires)

Quatre grandes familles de lecteurs de disquettes et DOS associés peuvent être distinguées dans le monde ORIC:

A) Le MICRODISC d' Oric Products International pour lequel 4 grands DOS ont été développés: ORIC DOS V1.1, RANDOS, XT DOS/XL DOS et SEDORIC

B) Le Jasmin de TRAN avec le TDOS/FTDOS.

C) Divers autres lecteurs dont les plus connus sont ITL KATHMILL (BDDOS), CUMANA (SUPER DOS), OPELCO (ROMDOS) et PRAVETZ (BOBY-DOS).

D) Enfin, le TELESTRAT avec son STRATSED.

A - LE MICRODISC

A.1) ORIC PRODUCTS INTERNATIONAL ET ORIC DOS V1.1

Eté 83: "Nos lecteurs MICRODISC sont en phase de finalisation et devraient être mis en production pour commercialisation vers septembre-octobre 83" (Peter Harding, directeur commercial de Oric Products International, cité, page 18, dans "ORIC, l'histoire sans fin" de Jonathan Haworth (noté par la suite dans cet exposé par la lettre H, suivie du numéro de page). "Le lecteur MICRODISC et le modem sont en bonne voie et pourraient sortir en septembre" (Paul Kaufman, directeur général de Tansoft, H19). De fait, dans Micr'Oric 2 (Automne 83) on trouve cette première publicité ORIC-1: "Bientôt un micro-lecteur de disquettes Oric" (page 3) et un concours, avec parmi les lots "Un lecteur de micro-disquettes" (page 62) (notez les variantes dans l'appellation et ce n'est pas fini!). Et dans le numéro suivant (Micr'Oric 3, Hiver 83-84, page 3), une publicité ORIC-1 signale "Stockage sur lecteur de disquettes 3" ORIC MICRO DRIVE extensible à 4 unités (mais toujours pas de photo). Dans le même numéro, la première publicité ATMOS "Le Nouveau Venu" montre un MICRODISC noir et rouge (page 65). Encore plus loin dans le même numéro de Micr'Oric (page 67), une autre publicité ORIC-1 montre la photo d'un ORIC MICRO DRIVE aux couleurs de l'ORIC-1 sur lequel on peut lire "ORIC-1" et en dessous ORIC MICRO DISC (en 3 mots) et conseille "Signalez votre réservation dans le bon de commande", mais sans indication de prix. Il semble que ce drive pour ORIC-1 n'ait jamais été commercialisé (H27). En effet, un prototype sans indication de prix est présenté au "Which Computeur Show" du 17 janvier 84, en même temps que l'ATMOS (H25). Pire, "le 4 février 84, ORIC organise une présentation à la presse du nouveau MICRODISC... et s'empresse de l'annuler" (H27).

En fait les premiers MICRODISC ne seront disponibles qu'au printemps 84 (environ 3000F) et souffriront d'incompatibilité matérielle avec l'ATMOS auquel ils sont destinés et qui apparaît en même temps (cf. la fameuse publicité "Maintenant, allez-y!"). Avec Micr'Oric 4 (Printemps 84, pages 3 et 62), Théoric 1 (avril 84, pages 10 et 32) et Oric Owner 7 (avril-mai, H27), les publicités se succéderont, ainsi que les appellations (ORIC MICRO-DISC, MICRODISQUES ORIC etc.). Les promesses se multiplient: double face, double densité (qui ne verra le jour que chez Eurêka) dont certaines sont assez fantaisistes: capacité 160 Ko par face, débit 250 Ko/s ou encore 640 Ko formatés, une seule tête, simple densité. Et que dire de “Evolution possible à 4 lecteurs 80 pistes / face, 3" ou 5,25", **mais le premier lecteur doit être un 3"”** (?!) . Voir notamment l'interview de Paul Jonhson et Terry Shurwood dans Théoric 1, page 10 qui annoncent entre autres que les nouveaux lecteurs de disquettes sont des 3,5" Hitachi (!) et où l'on apprend quand même que le connecteur est au standard Shuggart.

La première référence à un DOS pour MICRODISC, non nommé mais il s'agit sans doute du DOS V1.1 ou d'une version antérieure (V1.0?), se trouve dans un court article de Micr'Oric 4 (Printemps 98), page 62: "Le système d'exploitation est fourni sur disquette avec 17 fonctions, un mode d'emploi et des exemples à l'écran." Liste des 17 fonctions: !BACKUP, !CLOSE, !COPY, !DEL, !DIR, !DRV, !FORMAT, !GET, !LOAD, !OPEN, !PROTECT, !PUT, !RECALL, !REN, !SAVE, !STORE et !SYS.

Selon une autre source, un gros article de Théoric 2 (juillet 84, page 26 à 30), cinq de ces commandes sont orthographiées autrement (!DELETE, !DIRECTORY, !DRIVE, !RENAME et !SYSTEM). S'agit-il d'une autre version? Aucune référence précise n'est donnée. Mauvaise nouvelle: on apprend que le MICRODISC coûte plus cher que prévu, soit 3600F.

Cet article précise qu'au boot on a le choix entre 2 options !HELP et !DEMO. HELP donne un résumé de chaque commande du DOS, lequel occupe 45 secteurs et réside dans le fichier SYSTEM.DOS. Les disquettes sont simple face et formatées à raison de 40 pistes de 16 secteurs par face. Un secteur semble réservé pour le boot et un secteur pour le directory (nom de chaque fichier et adresse du premier secteur du fichier). Un autre article dans Micr'Oric 6 (Automne 84, page 35) donne de nombreuses informations supplémentaires. Pour faire extrêmement bref, le fichier SYSTEM.DOS est d'abord chargé en RAM (de #7400 à #A030) et exécuté en #A000, ce qui transfère le DOS en RAM overlay de #D400 à #FFFF. Si vous voulez en savoir plus, consultez cet excellent article de 7 pages intitulé "BONJOUR LES MICRODIQUES" par Fabrice Broche.

Certaines commandes ont des options, par exemple l'option MERGE pour COPY ou l'option ",N" pour LOAD. Pour sauver sous le même nom, il faut d'abord renommer ou supprimer l'ancien fichier. Il existe une liste de 29 erreurs possibles. Après chargement du DOS en RAM overlay, le système recherche, charge et lance BOOTUP.COM. Particularité amusante les fichiers ayant l'extension .COM peuvent être chargés sans le "!LOAD", simplement avec !nom_du_fichier (Micr'Oric 7, page 39). Certains utilitaires sont livrés sur la disquette Master, par exemple OLD.COM.

L'ensemble ATMOS+MICRODISC cause bien des soucis, non seulement du point de vue matériel (notamment parce que le signal d'horloge de l'ATMOS est anémique) que du point de vue logiciel: vecteur "!", HIMEM (Micr'Oric 7, page 38), LLIST, programmes LM (d'une part, après le boot, c'est la RAM overlay qui reste validée, salut les appels à la ROM, d'autre part le RTS final plante, voir Micr'Oric 5, page 35), variables système mal initialisées (telle la longueur de ligne écran mise à 80 au lieu de 40, voir Micr'Oric 5, page 34). Le DOS V1.1 a fait l'objet de plusieurs utilitaires, pour remédier à ses manques ou à ses bogues, notamment ceux de Denis Sebbag. DISK-SEARCH est une sorte de UNDELETE pour récupérer les fichiers accidentés (Micr'Oric 5, Eté 84, page 21). INITIALISATION est une sorte de menu

(Micr'Oric 5, page 23). Enfin et surtout, SUPER DOS, qui permet d'éditer l'ORIC DOS V1.1 et d'en faire un ORIC DOS V1.1S (Micr'Oric 5, page 24). Le nouveau fichier SYSTEM.DOS occupe 46 secteurs (#7300 à #A030, exécution en #A000). La capacité des disquettes est portée de 160 à 176 Ko par face soit 44 pistes de 16 secteurs au lieu de 40 pistes. Ceci grâce à la nouvelle commande !CONF, moyennant une bogue: après un BACKUP, il faut faire un POKE#500,#0. Le nouveau DOS accepte les noms de fichiers contenant un mot clé du BASIC (par exemple Z**OR**GON).

Théoric propose dans son numéro 6 de mars 85, page 47 à 50, un article intitulé "Analyse de disquettes", avec le programme ANADIS, qui permet d'explorer les disquettes du DOS V1.1. On y trouve de nombreuses indications. Voir aussi plus loin les articles de Fabrice Broche consacrés à une comparaison entre le DOS V1.1 et l'XL DOS ("Domptez votre MICRODISC", Théoric 8, mai 85, pages 40 à 43 et Théoric 9, juin-juillet 85, page 45 et 46).

Comme dans le cas des autres DOS pour ORIC, l'ORIC DOS V1.1 a probablement connu des évolutions et différents numéros de version (au minimum, il existe des versions 1.1 et 1.13).

Dans Théoric 4 (décembre 84, page 3), un nouveau DOS ORIC est annoncé, qui serait peut-être échangé gratuitement contre le DOS V1.1, mais toujours sans aucune référence précise. Il s'agit vraisemblablement du RANDOS. Notez qu'une certaine confusion règne alors, liée à la sortie quasi simultanée du XT DOS (voir plus loin) chez Micro Programmes 5, au prix annoncé de 450F, toujours sans aucune référence précise, sinon qu'une "capacité de 210 Ko/face" avec une "vitesse réelle de chargement de 10,5 Ko/s".

A.2) ORIC PRODUCTS INTERNATIONAL ET RANDOS

Micr'Oric 7 (février 85) titre "LE STRATOS ET LE RANDOS REVELES". Rappelons que Micr'Oric, c'est ORIC FRANCE, alias A.S.N. le fameux importateur. ORIC est toujours anglais et ces annonces sont le fait d'ORIC PRODUCTS INTERNATIONAL. A l'occasion du Salon Informatique de Francfort, le 1er février 85, le STRATOS est officiellement présenté (H37). Dès le lendemain, le 2 février, ORIC PRODUCTS INTERNATIONAL est mis en liquidation. Le STRATOS ne sera jamais commercialisé. Par contre RANDOS, le nouveau DOS, a échappé de peu à l'oubli total. Ce dos semble présenter des qualités **exceptionnelles** et il aurait probablement connu un grand succès, s'il n'était arrivé au mauvais moment. Notez qu'il fut proposé au prix de 80F, sur présentation de la facture du MICRODISC (Micr'Oric 8, page 44).

L'examen d'une disquette "master" RANDOS est particulièrement instructive. on y trouve dans le premier secteur les mentions "RADOS" et "Oric DOS V1.1", dans le secteur 7 de la piste 0 "Copyright (c) 1984 and property of" (le reste à été surchargé) et dans le secteur 6 de la piste 2 "**RANDOS V1.0.1 (c) ORIC 1983**". Tout cela fait assez désordre. On pourrait même penser que le RANDOS (RADOS?) a été développé en premier (1983) puis simplifié dans l'urgence en DOS V1.1 (1984) et enfin achevé et sorti en RANDOS (1985).

Revenons à Micr'Oric 7, page 36, où se trouve un court article intitulé "LE RANDOS D'ORIC", qui commence ainsi: "Dans les premiers mois de 1985, ORIC proposera son nouveau DOS appelé RANDOS et dont voici quelques caractéristiques (d'après une documentation écrite)". Pour résumer l'essentiel, ce DOS **gère des sous-répertoires**, ce qui est un cas unique dans le monde ORIC. Voilà qui pourrait faire le bonheur de ceux qui sont en train de développer une interface IDE pour l'ORIC, avec l'arrière pensée d'y installer un disque dur! Cette structuration du directory se fait grâce aux commandes !MAKE (pour créer un sous-répertoire) et !CHANGE (pour passer d'un répertoire à l'autre).

Le RANDOS possède également les commandes suivantes:
!BACKUP !BUILD !CLOSE !COPY (avec de nombreuses options) !CREATE !DEL !DEMO !DIR !DRV !ERROR !EXTEND !FILES !FILENAME !FORMAT !GET !LOAD !OLD !OPEN !OPTION BYTE !PROT !PUT !RECALL !REN !SAVE !SET !STORE !TYPE !WILDCARD. Notez que RANDOS dispose d'un accès disque octet par octet, ce qui peut être particulièrement utile pour le traitement de texte et pour les bricoleurs du soft. Notez aussi que !STORE et !RECALL permettent le transfert de données d'un programme à l'autre. "En matière d'information sur les erreurs, 42 messages différents peuvent être envoyés". Enfin, comble du bonheur, "le mode d'emploi de RANDOS apporte aussi la localisation et la description des routines utilisées".

Un autre article sur RANDOS, dans Théoric 7 (avril 85, page 57), souligne que le formatage est différent de celui du DOS V1.1, mais qu'il existe un utilitaire de transfert. L'article regrette l'absence d'indication de la taille des fichiers, mais souligne les innovations majeures (voir ci-dessus).

Comble de malchance, le RANDOS, successeur officiel du DOS V1.1, a été lancé au moment où Micro Programmes 5 sortait le XT DOS (alias XL DOS?) de Fabrice Broche et Denis Sebbag (voir plus loin). Toutefois, on a du mal à comprendre pourquoi EUREKA, le repreneur d'ORIC, a été obligé de développer un nouveau DOS (SEDORIC, du même tandem Broche-Sebbag) et surtout pourquoi SEDORIC n'a pas intégré les commandes les plus originales du RANDOS.

En fait beaucoup de questions se posent à propos du RANDOS (auteurs, parenté avec ORIC DOS V1.1, fiabilité, etc.) et nous nous proposons d'essayer d'y répondre dans un proche avenir. Toute information pouvant nous être utile sera la bienvenue (disquette et mode d'emploi d'origine par exemple). Lors du boot, RANDOS affiche toujours V1.0, mais l'existence de différentes versions ne peut être exclue.

A.3) MICRO PROGRAMMES 5 XT DOS ET XL DOS

Première allusion à un nouveau DOS pour MICRODISC dans Théoric 4 (décembre 84, page 7), commercialisé par Micro Programmes 5. Il comporte un BASIC étendu, les disquettes sont formatées à raison de 210 Ko/face. Le débit est de 10,5 Ko/s. Ce DOS n'est pas nommé, il s'agit vraisemblablement du XT DOS, qui est probablement la première version du fameux XL DOS.

Dans Micr'Oric 7 (février 85, pages 34 et 35), se trouve un article sur le nouveau XT DOS, qui commence ainsi: "Depuis fin 1984, on peut se procurer le XT DOS de F. BROCHE et D. SEBBAG, publié par Micro Programmes 5. La disquette fournie contient un mode d'emploi fort clair et d'accès très aisé". L'article détaille ensuite 5 groupes de commandes: (1) Travail sur disque, (2) Aide à la programmation, (3) Extension BASIC, (4) Système et (5) Gestion des fichiers (de type Matrice, Séquentiels, Accès direct, Chaîne et Disque). Voici la liste de ces commandes (avec entre parenthèses le groupe de commandes correspondant et éventuellement un petit commentaire):
&( ) (5), ACCEPT (3, saisie de texte formatée), ANGLE (3), BACKUP (1), BOX (3), CLI (4, autorise les interruptions clavier), CLOSE (5), CODE (3, codage chaînes BASIC), COPY (1), DEL (1), DELETE (2), DIR (1), DISK (5), DRV (1), EXECUTE (3, exécution chaînes BASIC), FDEL (5), FEND (5), FIELD (5), FILE (4, fichier par défaut), FJUMP (5), FORMAT (1), FSTART (5), FUNC (4, en liaison avec utilitaire DKEY), INIT (1), LINE (3), LOAD (1, avec 4 options), LSET (5), MERGE (2), MLOAD (5, recharge les tableaux), MSAVE (5, sauve les tableaux), NUM (2), OFF (4, rend le "!" facultatif), OLD (2), ON (4, rend le "!" obligatoire), OPEN "D" (5, ouvre un fichier disque), OPEN "L" (5), OPEN "R" (5), OPEN "S" (5, ouvre un fichier chaîne), PRINTER (4), PROT (1), PUT (5), REN (1), RENUM (2), RESET (4), RESTORE (3), ROT (3), RSET (5), SAVE (1, avec 4 options), SEI (4, supprime les

interruptions clavier), SWAP (3), SYS (1), TAKE (5), UPDATE (1).

La première mention de XL DOS semble se trouver dans Théoric 6 (mars 85, page 18). Pour 450F, Micro Programmes 5 fournit une disquette (protégée, non copiable, mais remplaçable en cas de malheur) contenant un manuel intégré. Cette disquette est accompagnée d'une simple feuille de présentation, sans manuel papier. XL DOS se charge en RAM overlay, utilise la page 4, différencie les minuscules des majuscules. Il est 2 fois plus rapide que le DOS V1.1 en lecture et 5 fois plus en écriture. Ce DOS offre un BASIC étendu, une gestion de la touche FUNCT, une gestion du BRK, de nouvelles commandes: ACCEPT, ANGLE, CLI, LINE, RESTORE N, ROT, SEI et SWAP, Le '!' n'est pas nécessaire. Les disquettes comportent 44 pistes de 19 secteurs (plus de 200 Ko par face). Les disquettes formatées avec le DOS V1.1 sont compatibles sauf pour BACKUP, car les disquettes n'ont pas le même format.

Dans Théoric 8 (mai 85, pages 40 à 43) et Théoric 9 (juin-juillet 85, page 45 et 46), Fabrice Broche signe deux articles intitulés "Domptez votre MICRODISC" dans lequel il donne, en parallèle pour le DOS V1.1 et pour XL DOS, des informations sur le contrôleur de disquette, son EPROM, les registres du WD 1793, les routines RWTS, les variables système, les principales routines de ces deux DOS, l'organisation des disquettes (secteur en-tête, secteur de catalogue, secteurs programme) et l'organisation des fichiers. On apprend entre autres, l'existence d'une table de vecteurs située en #D400 en RAM overlay pour les DOS et translatée en #E000 dans l'EPROM du contrôleur. XL DOS permet d'étendre sans limite le vocabulaire et ceci sans le '!'.

On sent que SEDORIC, du même tandem BROCHE-SEBBAG n'est pas loin. Ce premier DOS de Micro Programmes 5 a bel et bien existé en tant que XT DOS (par exemple Michel Zupan dans Théoric 13, page 15 signale que son programme tourne sur MICRODISC avec le DOS V1.1 et RANDOS mais nécessite un rétablissement des vecteurs d'interruption d'origine pour XT DOS et XL DOS). Mais par la suite le XL DOS lui a succédé. S'agit-il d'un simple changement de jaquette ou d'une version déboguée/améliorée? Nous ne le savons pas, n'ayant eu entre les mains que le XL DOS et non le XT DOS. Le mode d'emploi sur disquette de XL DOS a été clairement adapté du celui du XT DOS, puisque à trois reprises les correcteurs ont laissé passé la mention XT DOS au lieu de XL DOS.

L'examen du premier secteur des disquettes XL DOS révèle les copyrights suivants: "XL DOS V 0.6 par D.Sebbag et F.Broche © MP5" et "XL DOS Oric Basic étendu Version 0.6 par F. Broche & D. Sebbag" (notez l'ordre inversé des auteurs!). On trouve encore dans le secteur 5 de la piste 0: "XL DOS © Micro Programmes 5". Nous ne savons pas s'il existe d'autres versions, mais c'est peu probable, car le successeur direct de l'XL DOS semble bien avoir été SEDORIC.

En résumé, le XL DOS, maillon intermédiaire entre le DOS V1.1 et SEDORIC, n'a jamais connu le succès, probablement à cause de son prix, du fait qu'on ne pouvait faire de copie de sauvegarde et surtout à cause de l'arrivée de SEDORIC sur le marché. De toute façon, sans la mise en liquidation d'Oric Products International, il aurait eu du mal à s'imposer face au RANDOS sortit au même moment et dont les qualités indéniables n'auraient pu que s'améliorer.

A.4) EUREKA / ORIC INTERNATIONAL ET SEDORIC

Juin 85, ORIC change de mains et passe dans celles de M. Tallar (Editorial de Théoric 9, page 5). La photo du patron de S.P.I.D. (Société Prospective Internationale de Distribution), alias Eurêka, alias ORIC International, s'étale en couverture de Théoric numéro 10 (juillet-août 85). M. Tallar recrute Fabrice Broche qui se met immédiatement au travail. Théoric 36 (novembre 87, pages 8 et 9), publie une interview

de ce dernier: "En juin 1985, je rencontre J.C. Tallar, PDG d'ORIC nouvellement français. J'entre chez ORIC en août pour finir en compagnie de Denis Sebbag le SEDORIC". En septembre 85, première publicité annonçant SEDORIC, la première grande nouveauté d' Eurêka (voir Théoric 12, page 2). La rubrique "Nouvelles" de Théoric 14 (novembre-décembre 85, page 8), annonce la sortie d'un nouveau MICRODISC 3" (simple face), accompagné de SEDORIC (liste des 90 commandes et des principales caractéristiques, notamment existence d'un utilitaire de conversion pour relire les anciennes disquettes). Un banc d'essai se trouve en page 10 et 11 du même numéro. L'ensemble ATMOS français, MICRODISC et SEDORIC est alors pleinement fonctionnel.

Deux articles de F. Geothalls et F. Taraud, "En savoir plus sur le SEDORIC" dans Théoric 19 (avril-mai 86, pages 34 à 38) et "SEDUTIL, c'est utile", Théoric 21 (juin-juillet 86, pages 30 à 34) feront la joie des curieux. On y trouve des outils et des informations indispensables à tout oricien sérieux.

Une comparaison des commandes de SEDORIC et de FTDOS peut être trouvée dans un article de D. Vasiljevic (Théoric 28, février 87, page 17 à 20). Il ne nous semble pas nécessaire de donner la très longue liste des commandes de SEDORIC, ce DOS étant toujours largement utilisé.

Dernier avatar du MICRODISC: Oric International annonce la sortie (enfin!) d'un MICRODISC double face pour 2690F (Théoric 32, juin 87, rubrique "Nouveautés", page 8). En fait, il y aura un épisode supplémentaire: "Curieusement, le syndic en charge des affaires d'ORIC continue la vente. Début 88, un lecteur 31/2" est proposé aux derniers acheteurs...jusqu'au début du mois de mai, date à laquelle la boutique de la rue Victor Massé ferme définitivement ses portes" (H49).

SEDORIC connaîtra plusieurs versions, les plus célèbres étant la version 1.006 du 01/01/86, suivie bien plus tard par les versions 2.x et 3.0.

B - LE JASMIN

B.1) JASMIN et TDOS

Théoric 2 (juillet 84, pages 9 et 14 à 16) révèle le lecteur 3" Jasmin de la Société TRAN disponible au prix de 3590F. Ce lecteur simple tête est également proposé en version DUO (un master et un slave) pour 5890F. L'alimentation est incorporée dans les deux cas. La disquette 3" vierge vaut 65F. Dans Théoric 3 (septembre 84, page 59), le prix de ces lecteurs sont en légère hausse et la publicité est étendue à un lecteur double face (4390F) ainsi que sa version DUO (6990F)!

Ce lecteur est livré avec une disquette TDOS (pour TRAN DISK OPERATING SYSTEM) comptant "plus de 35 instructions". Il se charge en RAM overlay et permet de formater 178,5 Ko par face soit 357 Ko par disquette. Il existe deux versions de TDOS: pour ORIC-1 sur la face A de la disquette et pour ATMOS sur la face B. Le système boote grâce à une EPROM située sur la carte contrôleur. Les 35 instructions sont les suivantes:

!APPEND !CAT !CLOSE !COPY !DEL !DEMOUNT !DNAME !EROFF !ERRGOTO !ERSET !FORMAT !HSCREEN !INIT !INST !JUMP !LCAT !LECT !LOAD !LOCK !LSCREEN !UNLOCK !MASTER !MERGE !CREATE !MLOAD !LING !MOUNT !MSAVE !OPEN !RENAME !REWIND !SAVE !SEARCH !TKD !WHERE !WRITE

TRAN propose dans ses publicité un livre de Beaufils et Arnaud, intitulé "Le TDOS et ses fichiers pour ORIC-1 et ATMOS" au prix de 150F, qui sera présenté dans Théoric 8 (mai 85, page 11) à la rubrique "Biblioric".

Diverses corrections ou améliorations du TDOS ont été proposées. Cela semble commencer dans Théoric 6 (mars 85, page 54) par un entrefilet signé "un ami qui vous veut du bien. Le même ami (Guy Hermann) persiste dans Théoric 8 (mai 85, page 47), avec un article intitulé "Modifier le TDOS", dont on trouvera la suite dans Théoric 12 (septembre-octobre 85, page 55).

Nous ne connaissons pas le(s) numéro(s) de version du TDOS. Il a probablement évolué à plusieurs reprises. Par exemple, Théoric 4 (décembre 84, page 50) fait mention d'une version V2-26.w du TDOS et signale l'arrivée prochaine d'une nouvelle version "avec une vitesse de transfert 17 fois plus rapide"... probablement le FTDOS.

B.2) JASMIN et FTDOS

Annonce d'un nouveau DOS pour Jasmin "17 fois plus rapide que l'ancien", dans Théoric 5 (février 85, pages 54 et 55), le FTDOS V3-2 (pour Fast Tran Disc Operating System). Le numéro de version (3-2) indique qu'il s'agit en fait d'une nouvelle mouture du TDOS. De plus, le FTDOS, comme avant lui le TDOS existe en deux versions, ORIC-1 et ATMOS. Une mise à jour gratuite est proposée en magasin ou par courrier chez TRAN pour 100F (disquette vierge plus port) (Théoric 6, page 53). Ce DOS permet notamment de lire et d'écrire directement sur la disquette secteur par secteur et comporte 4 instructions supplémentaires par rapport au TDOS: !WS (pour Write Sector), !RS (pour Read Sector), !DS (pour Delete sector) et !HELP (qui charge les fichiers ayant l'extension .SCR). Le FTDOS comporte "50 instructions indispensables pour les applications de gestion et scientifiques et plus de 5 utilitaires"!

Le copyright est ainsi libellé:
T.R.A.N. DISK OPERATING SYSTEM V 3-2
© 1984 TECHNOLOGIE RECHERCHE ET APPLICATIONS NOUVELLES
83130 - FRANCE

Un article de Hervé Janod dans Théoric 25 (novembre 86, pages 30 à 35) est consacré au FTDOS V3-2. On y trouve des informations sur l'organisation des disquettes. Elles sont formatées en 41 pistes de 17 secteurs de 256 octets, deux secteurs y sont réservés: le secteur système (piste 20, secteur 1) où se trouve la bitmap et le premier secteur de directory (piste 20, secteur 2). En outre, la disquette peut contenir le DOS qui se trouve au début de la disquette et qui occupe un nombre variable de secteurs selon qu'il s'agit du FTDOS pour JASMIN "1" (62 secteurs) ou du FTDOS mixte pour JASMIN 2 et 2-PLUS. Le FTDOS comporte 42 fonctions plus les 4 utilitaires FORMAT, BKP, COPY1 et TKD. Ces commandes sont: !APND !CAT !CLOSE !COPY !CREATE !CUT !DEL !DEMOUNT !DNAME !EROFF !ERR !ERSET !FS !HELP !HSCR !INIT !JUMP !LCAT !LOAD !LOCK !LSCR !MASTER !MERGE !MLOAD !MOUNT !MSAVE !OPEN !RENAME !REWIND !RS !SAVE !SEARCH !START !TAKE !UNLOCK !UNSTART !WHERE !WL !WRITE !WS !WUL. Le rôle de ces commandes est indiqué dans un article de D. Vasiljevic (Théoric 28, février 87, page 17 à 20) où elles sont comparées aux commandes équivalentes de SEDORIC. On y trouve en plus !ERR GOTO et !DS, soit 43 commandes et 4 utilitaires.

Curieusement, la commercialisation des lecteurs double face a entraîné la sortie d'une variante du FTDOS, le FTDOS-DT, dans lequel la disquette est considérée comme ayant une seule face de 82 pistes (Théoric 5, page 54).

Un court article accompagné du listing du directory de la disquette master (version ATMOS) sur le FTDOS parait dans Théoric 7, pages 30 et 31. Il signale la correction des bogues qui affectait les commandes !MERGE, !CUT, !ERSET, !ON ERR GOTO et !LOAD, ainsi que l'apparition des nouvelles commandes: !FS (permet de connaître le premier Free Sector de la disquette), !RS, !DS et !WS (voir ci-dessus). Ces quatre nouvelles instructions sont un peu artisanales, puisqu'il faut faire des PEEK et POKE pour les utiliser!

Là encore, diverses corrections ou améliorations du FTDOS ont été proposées. Par exemple un article de Théoric 12 (septembre-octobre 85, page 56), propose une amélioration de la commande !START "palliant le défaut de lancement aléatoire d'un programme en langage machine" (en passant, nous apprenons l'existence de cette commande). Un utilitaire est proposé par Guy Hermann, dans Théoric 13 (octobre-novembre 85, page 53), permettant de lire les disquettes MICRODISC (il s'agit du DOS V1.1) sur Jasmin (il s'agit du FTDOS). Guy Hermann proposera l'utilitaire inverse dans Théoric 16 (janvier-février 86, page 52). Entre temps, SEDORIC est sorti et propose l'utilitaire CONVERT (voir manuel SEDORIC et Théoric 14 de novembre-décembre 85, page 10). Ce dernier utilitaire pour SEDORIC permet de relire les disquettes des DOS V1.1, XL DOS et TDOS.

B.3) JASMIN 2 et NOUVEAU DOS

Dans Théoric 7, page 35, première annonce d'un nouveau lecteur 3" de chez TRAN le JASMIN 2, compatible avec le précédent. Avec ce lecteur double tête "on accède directement aux 82 pistes" (357 Ko formatés). Prix de lancement 3490F. La première publicité pour ce Jasmin 2 s'étale en dernière page de ce Théoric 7. On apprend qu'il existe aussi en version double lecteur pour 5390F (prix corrigé en hausse à 5980F, dans la publicité suivante, Théoric 8, page2). Un article est consacré au Jasmin 2 dans Théoric 9 (juin-juillet 85, page 50). Par rapport au Jasmin "1", le nouveau produit est présenté comme offrant "l'avantage d'être un lecteur double tête. Cela signifie que vous n'aurez plus besoin de retourner la disquette dans le lecteur et que vos fichiers pourront s'étaler sur deux faces". Ceci semble curieux, puisque la première gamme de lecteurs Jamin proposait déjà des lecteurs double tête (au prix de 4390F en septembre 84 et même à 6990F en version DUO!).

Le Jasmin 2 est doté d'un nouveau DOS, "rapide comme le FTDOS", **qui est maintenant compatible ORIC-1 et ATMOS**. Aucune indication sur le nom de ce DOS, vraisemblablement une nouvelle version du FTDOS. André Guichardon propose dans Théoric 37 (décembre 87, page 36 et 37) une modification du DOS du Jasmin 2 pour déplacer la page 4, mais ne donne pas d'indication de version.

Face à l'offensive d'Eurêka avec son ORIC français, son nouveau MICRODISC et son fameux SEDORIC, la société TRAN se sent obligée de lancer son JASMIN 2-PLUS. Voici un extrait de leur publicité dans Théoric 14 (novembre-décembre 85, page 31): "La Société TRAN a mis au point JASMIN 2 - PLUS : nouvelle version du fameux JASMIN 2, avec circuit 'pré-diffusé' permettant une forte intégration des fonctions du contrôleur de disquette, d'où renforcement de la fiabilité accompagnée d'une baisse de prix". Le prix en question chute en effet à 2690F au lieu de 2990F le mois précédent, le MICRODISC, lui, est à 2490F depuis septembre. Le JASMIN 2 - PLUS fonctionne aussi avec le FTDOS (version?).

C - AUTRES DRIVES ET DOS

Plusieurs autres systèmes ont été commercialisés. Il s'agit soit de systèmes complets (carte contrôleur, drive et DOS) soit de systèmes utilisant un DOS préexistant ou légèrement adapté, soit de simples lecteurs

esclaves (5,25" ou 3,5" en général). N'ayant eu entre les mains aucun de ces drives, ni aucun des DOS correspondant, nous ne pouvons donner que des indications sommaires et probablement sujettes à caution. En voici la liste chronologique (probablement incomplète):

1) Avril 1984 ITL Kathmill, lecteur BD500 accompagné du BDDOS.
2) Juillet 1985, Cumana et SUPERDOS 2.2 (dérivé de l'ORIC DOS V1.1)
3) Janvier 1986, M.S.E., premier lecteur 5,25"
4) Mars 1986 I.C.V., premier lecteur 3,5" esclave pour MICRODISC ou Jasmin
5) Mars 1986, Opelco et ROMDOS

C.1) ITL KATHMILL

Dans H27, on peut lire: "En avril également (1984), ITL Kathmill lance son lecteur Byte Drive 500 à l'étude depuis juillet de l'année précédente. Ce dernier est bien noté dans 'What Micro' pour son jeu d'instruction étendu mais critiqué parce que son DOS est logé sous la mémoire écran et risque donc d'entrer en conflit avec certains programmes. Le BDDOS a été écrit par Peter Halford (déjà auteur des routines cassettes de l'ORIC-1 et d'Oric Mon)." A part cette citation, aucune trace de ce Byte Drive DOS. A noter l'annonce du lecteur de disquettes BD500 dans Théoric 4 (décembre 94, page 8).

C.2) CUMANA

"Un événement qu'il convient de noter en juillet (1985) : Cumana, âme courageuse, lance son lecteur de disquettes ORIC au prix de 235£" (H41). Le SUPERDOS 2.2 de Cumana n'est autre que l'ORIC DOS V1.1 légèrement modifié.

C.3) MSE ET LE PREMIER DRIVE 5,25"

"...en janvier (1986) , preuve supplémentaire que l'ATMOS a été diffusé dans d'autres pays, une firme allemande de Düsseldorf, MSE, propose un lecteur 51/4" pour cette machine" (H45). Nous ne savons pas si ce drive était muni d'une carte contrôleur et si un DOS l'accompagnait.

C.3) I.C.V. ET LE PREMIER DRIVE 3,5"

La société I.C.V. "Revendeur agréé ORIC Eurêka", 130 route de Corbeil, 91360 Villemoisson-sur-Orge, propose un lecteur de disquette 3,5" esclave, double tête, double densité, compatible MICRODISC et JASMIN 2, alimentation intégrée, pour 1990F (Théoric 18, mars-avril 86, page 5). Ce drive est passé au banc d'essai dans le même numéro de Théoric, page 8.

Notons en passant qu'à de multiples reprises des drives 5,25" esclaves ont également été proposés tant pour MICRODISC que pour JASMIN. Pour exemple une publicité de VISMO dans Théoric 23 (septembre 86, page 49) pour un nouveau lecteur 5"1/4 sur ORIC à 1595F.

C.4) OPELCO

"Opelco est apparu sur le marché britannique en mars (1986), avec ses premiers mailings" (H47). "En novembre 1986, Opelco lance une nouvelle gamme de lecteurs de disquettes: un modèle simple à 184£ et un modèle double à 235£, avec deux variantes de DOS (H48). Dans le JEO-MAG 4 (juillet 1990) on peut

lire: "M. Steve HOPPS, directeur de la société Opelco, nous a contactés pour nous informer qu'il possédait un stock important de lecteurs de disquettes Master neufs qui n'attendaient qu'à trouver un foyer...le prix se fixera aux alentours de 1600F" (article de Vincent Talvas). Les caractéristiques du lecteur Opelco sont passées en revue: 3", boîtier métallique, alimentation incorporée, modèles simple ou DUO, simple ou double face, norme Shuggart, livré d'origine avec deux DOS: **ROMDOS** et RANDOS, "mais SEDORIC fonctionne également avec l'Opelco".

C.5) PRAVETZ

C'est grâce à internet que la communauté Oricienne a appris l'existence d'un drive PRAVETZ 8D, compatible avec le format de fichier et de disquette de l'APPLE][(5"1/4, simple face, simple densité, 140 koctets). En Juillet 1998, Ivan Naydenov de Sofia (Bulgarie) lançait sur oric@lyghtforce.com un appel au secours, parce que READDSK.EXE ne pouvait pas relire ses disquettes 5"1/4 ORIC "compatibles APPLE" pour les utiliser sous EUPHORIC.

C'est ainsi qu'il nous apprit qu'au milieu des années 80, commença en Bulgarie la fabrication de l'ordinateur "PRAVETZ 8D", un clone de l'ATMOS avec caractères cyrilliques et alimentation interne (la ROM cyrillique est disponible sur une page web de Fabrice). Mais ce PRAVETZ 8D ne disposait que d'un lecteur de K7, alors que ses confrères, les PRAVETZ 82, 8M et 8C, pavanaient avec leurs lecteurs de disquette et leur DOS 3.3. C'est alors qu'un ingénieur bulgare, Borislav Zahariev, adapta un lecteur "APPLE][" sur un PRAVETZ 8D (ATMOS). Le transfert de fichiers entre le monde APPLE et le monde ORIC pouvait se faire sans problème, pour le texte évidemment, les programmes nécessitant une adaptation. Le BOBY-DOS, écrit par Borislav Zahariev, est compatible APPLE. Les disquettes ont toutes les caractéristiques APPLE: même taille et emplacement des secteurs, même structure. Toutefois, si le "file type" utilise les mêmes bits que l'APPLE, les lettres désignant le type de fichier sont différentes (c'est uniquement un problème de visualisation par le DOS).

Rappelons que les drives APPLE sont au format MF (simple densité), alors que les drives ORIC et PC sont au format MFM. Les contrôleurs MFM des PC ne semblent pas savoir lire d'autre format que MFM, donc il n'est probablement pas possible d'adapter READDISK pour lire les disquettes du PRAVETZ. Mais le contrôleur du MICRODISC est capable de lire les formats FM. Il existe un bit de sélection, qui n'est géré par aucun des DOS ORIC. Selon Fabrice, il serait possible d'écrire une routine spéciale pour permettre aux ORIC de lire les disquettes APPLE/PRAVETZ.

D - LE TELESTRAT

D.1) ORIC PRODUCTS INTERNATIONAL ET IQ164/STRATOS

Très curieusement, Micr'Oric (revue officielle de ORIC FRANCE, alias l'importateur A.S.N.) présente, à quelques pages d'intervalle du même numéro (Micr'Oric 7) de février 85, le STRATOS (pages 9 à 11) et l'IQ164 (pages 43 à 45) avec les mêmes caractéristiques (encore heureux!). Information reprise par Théoric 6 (mars 85, page 14), qui avance un prix d'environ 4000F. En fait, en Angleterre, le nom STRATOS était déjà breveté et faute de mieux, la machine aurait gardé son nom de développement, le IQ164. Pourquoi s'intéresser à une machine qui ne fut jamais diffusée? En ce qui nous concerne pour l'instant, à savoir les drives et DOS, nous aimerions simplement reconstituer l'évolution jusqu'au STRATSED. Nous ne ferons aucun commentaire sur les autres possibilités de la machine, notamment sur

le mode d'affichage 26 lignes sur **80 colonnes** ou la haute résolution adressée bit par bit, avec attributs parallèles!

Première révolution, la machine reçoit son langage d'une cartouche et dispose d'un espace mémoire étendu grâce à un système de permutation de banques (en fait deux cartouches sont utilisables)."La cartouche fournie d'origine propose un BASIC super étendu, qui contient outre toutes les commandes du BASIC 1.1 de l'ATMOS, le système d'exploitation des MICRODISC (DOS) et 31 commandes supplémentaires dont nous donnons la liste plus loin". Citons encore: "La compatibilité avec l'ATMOS est assurée à 100% grâce à la commande "ATMOS"... tout simplement!". Passons sur une liste de promesses époustouflantes (par exemple futur système CP/M)...

Revenons sur les 31 commandes supplémentaires disponibles (auxquelles il faut encore ajouter celles du DOS proprement dit, dont on ne sait malheureusement rien):
ABS DRAW (commande graphique), ADRAW 3D (commande graphique), ATMOS (compatibilité 100%), AUTO, DELETE, DRAW 3D (commande graphique), DSET 3D (commande graphique), ECLOAD (CLOAD amélioré, avec contrôle de CRC), ECSAVE (CSAVE amélioré), ED (éditeur), ELLIPSE (commande graphique), ENGLISH, ENV, EVAL, FRENCH, GDIR (pour cartouche de jeu), GLOAD (pour cartouche de jeu), IRS 232 (port série), MOVE 3D (commande graphique), NOTE, PAINT (commande graphique), RENUM, SETFUN (touche FUNCT), SINPUT (port série), SLIST (port série), SPLOT (commande graphique), SPRINT (port série), SRECALL (communication téléphonique avec un autre STRATOS), XLOAD (communication téléphonique avec un autre STRATOS), communication téléphonique avec un autre STRATOS), XSTORE (communication téléphonique avec un autre STRATOS).

Coté drives, un contrôleur de disquette est intégré à la carte mère. Quatre lecteurs esclaves de format 3" ou 5,25" peuvent être branchés. Capacité 160 Ko/face. Interface SHUGART (Théoric 6, page 14).

D.2) EUREKA / ORIC INTERNATIONAL ET TELESTRAT

Théoric 36 (novembre 87, pages 8 et 9), publie une interview de ce Fabrice Broche: "Pour le TELESTRAT, l'aventure commence en novembre 1985 **et se finit, pour la programmation, en septembre 1986**. Le TELESTRAT est certes arrivé en retard, mais il représente tout de même 55 Ko d'assembleur optimisés, sans compter le travail de routine à ORIC. En 10 mois, ce n'est pas si mal. Le travail se répartit à peu près ainsi: 2 mois de TELEMATIC, 3 mois de TELEMON et STRATSED (toutes les routines système), 5 mois de HYPERBASIC..."

En fait, le TELESTRAT aurait dû arriver au début de 86, comme en témoigne Théoric 16 (janvier-février 86), dont la couverture s'orne de l'énorme photo d'un TELESTRAT. On trouve aussi, page 9, avec un article intitulé "ORIC: LA FAMILLE S'AGRANDIT!" également illustré d'une large photo du TELESTRAT. Selon une publicité sur la dernière page de couverture, la machine serait disponible chez VISMO au prix de 3990F (*prix indicatif au 31/12/85).

Le numéro suivant, Théoric 17 (février-mars 86), présente pages 8 et 9 le "TELESTRAT, le nouvel ORIC". Coté lecteur de disquette, on en est toujours au 3", probablement simple tête, puisque le MICRODISC double face n'est sortit chez ORIC International qu'en juin 1987. Aucune indication sur le DOS, si ce n'est la présence d'un "BASIC de plus de 250 instructions". On trouve toujours la même publicité VISMO en dernière page de couverture.

Le démarrage semble bien dur: aucun article n'est consacré à cette nouvelle machine, à part les publicités (Théoric 18 de mars-avril 86, page 5 et 60; Théoric 19 d'avril-mai 86, page 21; Théoric 20 de mai-juin 86, page 5 et surtout pages 29 à 34; Théoric 21 de juin-juillet 86, page 15; Théoric 22 de juillet-août 86, pages 15 et 25; Théoric 23 de septembre 86, pages 25 et 26). Limitons nos propos aux annonces concernant le drive et le DOS: on connaît seulement l'existence d'un HYPER-BASIC ("2 à 100 fois plus rapide") et du système d'exploitation STRATSED. Le TELESTRAT "dispose dès sa naissance de plus de 2000 programmes...Outre son BASIC, le TELESTRAT pourra également recevoir un langage C, un FORTH, un PASCAL...". Le MICRODISC est annoncé "double tête". La liste des instructions HYPER-BASIC et STRATSED est donnée dans Théoric 20, page 34 et elle est effectivement impressionnante. En fait le TELESTRAT n'est toujours pas disponible!

octobre 86, Théoric 24 titre 'Le TELESTRAT est là!". Cette fois ce n'est pas une blague, le TELESTRAT est enfin disponible, avec 9 mois de retard. C'est aussi le début d'une série de nombreux articles sur cette machine. Citons ceux de Fabrice Broche: "Le TELESTRAT: plus qu'un nouvel ORIC" de dans Théoric 24 (octobre 86, pages 19 à 25); "TELESTRAT: structure matérielle et logicielle" de Fabrice Broche dans Théoric 30 (avril 87, pages 12 et 13); "Trucs et astuces" (Théoric 31, mai 87, page 22); "Le brochage connecteurs" (Théoric 32, juin 87, pages 39 à 41, notez que F. Broche est plutôt discret sur la prise MIDI !); "Gestion des canaux" et "Trucs et astuces" (Théoric 33, juillet 87, pages 14 et 15) et "Structure des fichiers TELEMATIC" (Théoric 37, décembre 87, pages 10 et 11).

Enfin, une adaptation de SEDORIC V1.006 tournant sur TELESTRAT est proposée au prix de 490F, il s'agit du kit STRATORIC. Ce kit permet en outre d'émuler les ROM V1.0 et V1.1 et ,grâce à l'utilitaire CONVERT, de relire les disquettes DOS V1.1 et Jasmin (Théoric 31, mai 87, page 18).

Pour conclure, nous pourrions regretter, une fois du plus, les 9 mois de retard du TELESTRAT, qui ont été pour beaucoup dans la disparition d'ORIC, face à la concurrence féroce du moment. Sous la pression d'ORIC International, le pauvre F. Broche a dû avoir l'épée dans les reins aux cours de ces 9 mois, durant lesquels il a dû programmer comme un fou! La chute était inéluctable: tous les autres micros de cette époque sont tombés. Quant aux applications... il est encore temps de s'y mettre!

ANNEXE n° 24
Directories des disquettes patchées : SEDORIC V 3.006 & TOOLS V3.006

```
Drive A V3 (Mst)  SEDORIC V3.006 + P

ADDRESS  .COM  12P  ADDRESS  .DAT  30P
ADDRESS  .WIN   5P  ALPHA    .COM  14P
BDDISK   .COM  59P  BDDISK   .HLP   6P
BDDISKAC .COM  59P  CHKSUM   .HLP   6P
CONVERT  .COM  31P  DEMO     .COM 149P
EUPHORIC .BK6  65P  EUPHORIC .BK7  65P
GAMEINIT .COM  22P  K        .B1   33P
K        .B2  157P  KRILLYS  .COM   2P
KRILYS   .BIN   2P  MARC     .COM  98P
MENU     .COM   5P  MONAC1   .COM  10P
MONAC1   .HLP   6P  NIBBLE   .COM  27P
NIBBLE1  .HLP   6P  NIBBLE2  .HLP   6P
NIBBLE3  .HLP   6P  NIBBLERAY.COM  27P
ROMATMOS .COM  66P  ROMORIC1 .COM  66P
SECTMAP  .BIN   2P  SECTMAP  .COM   6P
SECTMAP  .DAT   5P  SEDORIC1 .KEY   3P
SEDORIC3 .FIX   6P  SEDORIC3D.KEY   3P
SEDORIC3N.KEY   3P  STAT     .COM   3P
STRAT3   .256 130P  V20      .COM   2P
V20      .PG1   5P  V20      .PG2   5P
V30NEWS01.HLP   6P  V30NEWS02.HLP   6P
V30NEWS03.HLP   6P  V30NEWS04.HLP   6P
V30NEWS05.HLP   6P  V30NEWS06.HLP   6P
V30NEWS07.HLP   6P  V30NEWS08.HLP   6P
V30NEWS09.HLP   6P  VERSION  .COM   6P
VISUHIRES.HLP   6P  WELCOME  .HRS  33P
PATCH    .002   4P  PATCH    .001   6P
PATCHHELP.001   6P  PATCHHELP.002   6P

1119 sectors free (D/80/16) 56 Files
```

```
Drive A V3 (Mst)   TOOLS V3.006 + P

ADDRESS  .COM  12P  ADDRESS  .DAT  30P
ADDRESS  .WIN   5P  ALPHA    .COM  14P
BDDISK   .COM  59P  BDDISK   .HLP   6P
BDDISKAC .COM  59P  CHKSUM   .HLP   6P
CONVERT  .COM  31P  DEMO     .COM 149P
EUPHORIC .BK6  65P  EUPHORIC .BK7  65P
GAMEINIT .COM  22P  K        .B1   33P
K        .B2  157P  KRILLYS  .COM   2P
KRILYS   .BIN   2P  MARC     .COM  98P
SECTMAP  .COM   6P  MONAC1   .COM  10P
MONAC1   .HLP   6P  NIBBLE   .COM  27P
NIBBLE1  .HLP   6P  NIBBLE2  .HLP   6P
NIBBLE3  .HLP   6P  NIBBLERAY.COM  27P
ROMATMOS .COM  66P  ROMORIC1 .COM  66P
SECTMAP  .BIN   2P  MENU1    .COM   5P
SECTMAP  .DAT   5P  SEDORIC1 .KEY   3P
SEDORIC3 .FIX   6P  SEDORIC3D.KEY   3P
SEDORIC3N.KEY   3P  STAT     .COM   3P
STRAT3   .256 130P  V20      .COM   2P
V20      .PG1   5P  V20      .PG2   5P
V30NEWS01.HLP   6P  V30NEWS02.HLP   6P
V30NEWS03.HLP   6P  V30NEWS04.HLP   6P
V30NEWS05.HLP   6P  V30NEWS06.HLP   6P
V30NEWS07.HLP   6P  V30NEWS08.HLP   6P
V30NEWS09.HLP   6P  VERSION  .COM   6P
VISUHIRES.HLP   6P  WELCOME  .HRS  33P
NIBFIX   .COM   4   MENU2    .COM   4
BOOT     .COM   6   BOOT     .HLP   7
BOOT     .PRN   8   CAT      .HLP   6
CAT      .COM   9   CDA      .SM   16
CDA      .COM   4   CDA      .HLP   6
CDS      .HLP   6   CDS      .COM   3
CF       .COM   9   CF2      .COM   8
CF2      .HLP   6   COPFORM2 .COM  12
COPFORM  .COM  12   CHERAMY1 .COM   7
CHERAMY2 .HLP   6   CHERAMY3 .HLP   6
CHERAMY3 .COM   7   CHERAMY1 .HLP   6
DTR      .COM   3   CMP      .BAS  13
CMP      .MAC   6   CMP      .HLP   6
CMPB000  .COM   3   CMP9000  .COM   3
CS       .COM   3   CS       .HLP   6
DISKCOMP2.COM  41   DISKCOMP2.HLP   6
DISKSPY  .COM  21   DISKSPY  .HLP   6
DTR      .HLP   6   CHERAMY2 .COM   7
EDITECRAN.RAM  10   EDITECRA3.HLP   6
EDITECRA1.HLP   6   EDITECRAN.MEN   6
EDITECRAN.CHS   4   EDITECRAN.COM  44
EDITECRA4.HLP   6   EDITECRA2.HLP   6
EXPLOSED .HLP   6   EXPLOSEDM.MAC  11
EXPLOSED .COM  83   HARDCOPYT.COM   2
HARDCOPYH.COM   2   HARDCOPYT.HLP   6
```

```
HARDCOPYH.HLP    6   HENNINOT .COM    7
HENNINOT .HLP    6   IO       .COM    2
IO       .HLP    6   HR       .IO     3
HE       .IO     3   FIC      .IO     2
H        .IO     3   SCROLLBIN.IO     2
SCROLL   .IO     2   SEBIN    .IO     2
SR       .IO     3   MENU     .IO     3
SRBIN    .IO     2   VISUFIC  .IO     3
SE       .IO     3   HERBIN   .IO     2
S        .IO     3   KLOADMOV3.HLP    6
KLOADMOVE.SM    10   KLOADMOV1.HLP    6
KLOADMOV2.HLP    6   KLOADMOVE.COM    3
MENU     .BIN    3   MONDH    .COM    3
MONDH    .HLP    6   MOVER    .HLP    6
MOVBAS   .HLP    6   MOVBAS   .SM     3
MOVER    .COM    2   MOVBAS   .COM    2
QUITAC   .HLP    6   QUITAC   .SM    13
QUITAC   .COM    3   SECTMAP  .HLP    6
SECTNUL  .HLP    6   SECTNUL  .COM    2
SEDCAT20 .HLP    6   SEDCAT10 .HLP    6
SEDCAT10 .BAS    2   SEDUTIL  .BM0    2
SEDUTIL  .BM1    2   SEDUTIL  .HCO    2
SEDUTIL  .COM   33   SEDUTIL  .UTS    2
SEDUTIL  .DMP    2   SEDUTIL  .HLP    6
SHASM0600.COM   37   SHASM0600.HLP    6
SHMON9380.COM   35   SHMON1380.COM   35
SHMON9380.HLP    6   SHMON1380.HLP    6
TBD1     .HLP    6   TBD      .COM    2
TBD2     .HLP    6   TBD1     .COM    6
TBD2     .COM    5   TBD3     .COM    6
TBD0     .COM    5   E        .COM    6
M        .COM    4   UTIL1    .HLP    6
UTIL1    .COM    7   UTIL2    .COM    7
UTIL2    .HLP    6   VDT      .COM    2
VDT      .HLP    6   VH       .COM    2
VH       .HLP    6   WARMATMOS.SM     9
WARMATMOS.HLP    6   WARMATMOS.COM    3
WARMSEDOR.HLP    6   WARMSEDOR.SM     9
WARMSEDOR.COM    3   MENU     .COM    5
PATCH    .002    4P  PATCH    .001    6P
PATCHHELP.001    6P  PATCHHELP.002    6P

*307 sectors free (D/80/17)184 Files
```

ANNEXE n° 25
Tables et figures

Table des variables systeme 9
BUF1 16
BUF2 17
BUF3 18
Message: DOS IS ALTERED! 23
Divers messages (Initialisation SEDORIC) 24
Dump de la page 4 25
MOVE descendant (par le début) 47
MOVE ascendant (par la fin) 49
Messages de la BANQUE n°2 . . . . . 58 à 60
Table de formatage (BACKUP) . . . . . 59 et 60
Structure d'une piste (BACKUP) . . . . . 62 et 63
Messages de la BANQUE n°3 88
Messages de la BANQUE n°4 102
Messages de la BANQUE n°5 . . . . . 121 et 122
Messages de la BANQUE n°6 . . . . . 125 et 136
Table de formatage (INIT) 135
Structure d'une piste (INIT) . . . . . 138 et 139
Copyright (BANQUE n°7) 146
Table des codes de touches 161
Table "KEYDEF" . . . . . 162 et 163
REDEF: 16 commandes re-définissables avec KEYUSE . . . . . 164 à 166
PREDEF: 16 commandes pré-définies (code 16 à 31) . . . . . 164 et 165
Mots-clés SEDORIC (codes 32 à 127) 167
Sous-table selon la première lettre du mot-clé SEDORIC 172
Table des adresses d'exécution des mots-clés SEDORIC 173
Table NOM et EXTENSION par défaut 174
Table de constantes diverses 174
Table de conversion QWERTY / AZERTY 174
Table de conversion ACCENT OFF / ACCENT SET 174
Table de constantes diverses 174
Variables réservées par le système 175
Messages d’erreur SEDORIC (zone CDBF) . . . . . 175 et 176
Autres messages SEDORIC (zone CEE7) . . . . . 177 et 178
Registres et commandes du FDC 1793 . . . . . 179 et 180
Rappel des codes de touche 225
Rappel de la table "KEYDEF" . . . . . 226 et 227
Jeu de caractères français dit "accentué" 324
Paramètres de LINPUT . . . . . 331 et 335

Commandes SEDORIC faisant appel à une BANQUE externe . . . . 315, 316 356 et 357
Commandes de gestion de fichiers “S”, “R” et “D” 375
Structure du "Pseudo-Tableau" FI 380
Table des vecteurs système (#FF43-#FFC6) 456
Table KEYDEF 468
Table REDEF et PREDEF . . . . 469 et 470
Emplacement de SEDORIC sur une disquette Master 16 secteurs/piste 485
Emplacement de SEDORIC sur une disquette Master 17 secteurs/piste 486
Emplacement de SEDORIC sur une disquette Master 18 secteurs/piste 487
Dump du premier secteur de disquette 489
Dump du deuxième secteur de disquette 489
Dump du troisième secteur de disquette 493
Exemple de secteur 1 de la piste #14 (20) 493
Exemple de secteur 2 de la piste #14 (premier secteur de bitmap) 494
Exemple de secteur 3 de la piste #14 (deuxième secteur de bitmap) 496
Exemple de Directory 496
Dump du descripteur de la BANQUE n°7 497
Exemples de descripteurs de fichiers simples . . . . 499 à 501
Exemples de descripteurs de fichiers “mergés” . . . . 501 à 503
Dumps illustrant la commande SAVE . . . . 505 à 510
Dumps illustrant la commande DEL . . . . 511 à 516
Listing de l’EPROM du MICRODISC . . . . 517 à 548
Commandes du FDC 1793 . . . . 549 à 559
Comparaison des formats de piste IBM et ORIC 551
Liste des logiciels "moniteur/assembleur/dé-assembleur" adaptés pour SEDORIC 563
Table des Mots-Clés SEDORIC . . . . 575 et 576
Table des codes de fonctions . . . . 577 à 583
Mémoire libre en RAM overlay 584
Routines d’intérêt général (ordre chronologique) . . . . 587 à 596
Routines d’intérêt général (par thèmes) . . . . 597 à 610
Directories de la disquette SEDORIC V3.006 patchée 001 et 002 623
Directories de la disquette TOOLS V3.006 patchée 001 et 002 . . . . 623 à 625
Tables et Figures . . . . 626 et 627
Tables des Matières . . . . 628 et 629

ANNEXE n° 26
Table des matières

Avant-propos 3
Comment lire ce livre 4
Nouveautés de la version 3.0 5

La RAM overlay 7
Analyse des commandes SEDORIC 7

Buffer 1 (BUF1) 16
Buffer 2 (BUF2) 17
Buffer 3 (BUF3) 18

BANQUE n°0 19
Initialisation SEDORIC 19
Source de la page 4 version ORIC-1 25
Source de la page 4 version ATMOS 25
Désassemblage de la page 4 SEDORIC 26

BANQUES interchangeables 30
BANQUE n°1 (adresse Cxxx**a**): RENUM, DELETE et MOVE 30
BANQUE n°2 (adresse Cxxx**b**): BACKUP 51
BANQUE n°3 (adresse Cxxx**c**): SEEK, CHANGE et MERGE 68
BANQUE n°4 (adresse Cxxx**d**): COPY 89
BANQUE n°5 (adresse Cxxx**e**): SYS, DNAME, DTRACK, TRACK, INIST, DNUM, DSYS, DKEY et VUSER 103
BANQUE n°6 (adresse Cxxx**f**): INIT 123
BANQUE n°7 (adresse Cxxx**g**): CHKSUM, EXT, PROT, STATUS, SYSTEM ,UNPROT et VISUHIRES 144

Début du NOYAU permanent de SEDORIC (#C800 à #FFFF) 161
Mots Clés SEDORIC 167

XRWTS Routine de gestion des lecteurs 179

Série d'appels à des sous-programmes en ROM 187
Routines SEDORIC d'usage général . . . . 197, 212 et 236
Routines principales de Ray McLaughlin 292

Entrée SEDORIC: recherche l'adresse d'exécution d'un mot-clé SEDORIC 199
Analyse d'un nom de fichier 203
Prendre un caractère au clavier (remplace EB78 ROM) 221

Commandes SEDORIC (avec quelques routines associées, d'usage général) . . . . 232 et 251
Commandes SEDORIC faisant appel à une BANQUE externe 356

Note sur les coordonnées colonne/ligne ORIC-1 / ATMOS / SEDORIC 363

Gestion de fichiers 374

Table des vecteurs système (#FF43-#FFC6) 456

Copyrights . . . . 6, 146, 458, 461, 465, 484 à 486 et 488

ANNEXES 460

ANNEXE n° 1: SEDORIC V2.0 461
ANNEXE n° 2: SEDORIC V2.0 463
ANNEXE n° 3: SEDORIC V2.0 465
ANNEXE n° 4: PATCH 001 475
ANNEXE n° 5: PATCH 002 478
ANNEXE n° 6: Que se passe t-il lors du boot ? 482
ANNEXE n° 7: Rappel de la structure des disquettes SEDORIC 484
ANNEXE n° 8: Que se passe t-il lors d'un SAVE ? 504
ANNEXE n° 9: Que se passe t-il lors d'un DEL ? 511
ANNEXE n° 10: Listing de l'EPROM du MICRODISC 517
ANNEXE n° 11: Le FDC 1793 549
ANNEXE n° 12: F.A.Q 560
ANNEXE n° 13: Exercices de passage ROM <--> RAM overlay 562
ANNEXE n° 14: Utilisation d'une commande SEDORIC sans argument (programme LM) 564
ANNEXE n° 15: Utilisation d'une routine en RAM overlay (programme LM) 565
ANNEXE n° 16: Utilisation d'une commande SEDORIC avec paramètres (programme LM) 566
ANNEXE n° 17: Les bogues de SEDORIC 569
ANNEXE n° 18: Mots clés SEDORIC 575
ANNEXE n° 19: Les Codes de Fonctions 577
ANNEXE n° 20: Futures extensions 584
ANNEXE n° 21: Routines d'intérêt général (par ordre chronologique) 587
ANNEXE n° 22: Routines d'intérêt général (par thèmes) 597
ANNEXE n° 23: Des drives et des DOS pour ORIC 611
ANNEXE n° 24: Directories des disquettes SEDORIC V3.006 et TOOLS V3.006 623
ANNEXE n° 25: Tables et figures 626
ANNEXE n° 26: Table des matières 628

NOTES PERSONNELLES